

Using STACK in Moodle: Guide

Writing questions in Moodle using the STACK question type is the only effective way of writing a question in Moodle, because of the level of detail, randomisation and use of Maxima commands which can be employed.

This document contains four parts:

1. What does each section in STACK mean?

This will introduce the layout of STACK questions, explain what each part of means, a very brief introduction to how it works and whether it will be important.

2. Walkthrough of a basic question

This is ideal if you're starting your first STACK question. It will take you through a very basic level of how to write a STACK question, and introduce you to the manner in which you should be writing questions.

3. Walkthrough of an advanced question

Similar to the basic question, this will guide you through an example question. However, this will introduce some more complicated and advanced concepts by using novel techniques in Maxima.

4. Hints and Debugging

This should be your go to place if you're experiencing any bugs with STACK. There are also some hints, if you're struggling to design questions as you intend to.

5. Basic Maxima Syntax

This section provides you with some of the most common Maxima functions you will use, with a brief explanation of how they work.

Here are some useful links that may be referred to through this document:



- The official STACK documentation:
<http://stack.bham.ac.uk/moodle/question/type/stack/doc/doc.php/>
 - This will have most of the information you'll need to know about STACK, if there's anything this document has missed.
- Maxima documentation: <http://maxima.sourceforge.net/docs/manual/maxima.html>
 - A very detailed list of functions in Maxima. Note that not all may be used in STACK.
- STACK forum: <https://moodle.org/mod/forum/view.php?id=752>
 - A very technical forum about STACK in Moodle. Perhaps it is best to report any problems to IT, but this is a noteworthy link

What does each section mean in a STACK Question?

Firstly, we should take a look at what all the boxes and sections mean when we start a question. Most are self-explanatory, but I shall be thorough and explain each one. Some I will refer to in greater detail later, as they are far more important, whereas others aren't worth even talking about as you'll never use them.

General

This section is about the setup of the question, regardless of any answer submitted.

- The **category** is simply whereabouts in the question bank directory the question will be saved. Don't worry about this for now, you can move the question about once it has been saved anyway.
- The **question name** is the name of the question, simple.
- The **question variables** are quite an important part. Everything in here is written in maxima. You will use this as a place to define important variables and perform mathematical calculations that you'll want to refer to later in the question text and solution. When used effectively, it'll be massively time saving and save complicated and messy code. You shall see it in action in the question walkthroughs.
- I've always left **random group** blank, it has little use.
- The **question text** is where you will write the question which the student will see. The formatting in general works like a word document, (e.g. line spaces in this box give line spaces displayed). However, all your typing and formatting is actually being translated to HTML, which can be viewed by one of the buttons. It has the ability to run LaTeX math modes at any point using $\backslash(\backslash)$ and $\backslash[\backslash]$, as well as referring to question variables and running maxima commands by using @ @. To use this box effectively, I strongly recommend reading both example questions. The buttons are self-explanatory with a few exceptions, which will be extremely useful:
 - When you type in the question box, it translates your formatting and such to HTML, and this button is where you can see it all the HTML code: .
 - Highlight some text and click this button: , and all of you're the HTML formatting will disappear on it.
- The **default mark** is how many marks you wish to make the question out of.
- The **specific feedback** is what is shown to the student the first time they submit an answer. The environment works in exactly the same way as the question text. I've been using this section to store "hints", but unless you're giving multiple attempts, this is a fairly useless environment.
- **Penalty** is the deduction of marks made if multiple attempts are allowed and they provide an incorrect attempt.
- **General feedback** is what is shown to the student once they either get the correct answer or run out of attempts. It's where you'd store the model solution. It uses the same environment as the question text and specific feedback.
- **Question note** is a summary of the question. It's used as an administrator's note containing enough information to recreate student's exact question, especially when random numbers are used.

- **Verify the question text and update the form** is used when you have typed more answer boxes in the question, and need the page to update so the correct amount appear in your options

[Input: ans1](#)

This section is about the details of the answer submitted by the student.

- **Input type** is the allowed type of answer they should provide. 99% of the time you'll use algebraic input, which contains numbers, functions, and expressions with variables.
- **Model answer** is what will show as the correct answer. 99% of the time, you'll want to create a question variable with the name "ta1" and put ta1 in here, for easier referencing.
- **Input box size** is the size of the answer box displayed to the student in the question.
- **Strict syntax** is not important, leave it as no.
- **Insert stars** is not important, leave it as "don't insert stars".
- **Syntax hints** can be useful if you want to force the students to answer in a certain way. Whatever you type in here will come up in the answer box already. For example, I've entered "(?)*i + (?)*j + (?)*k" in here before to force i,j,k notation instead of allowing the students to attempt matrix notation.
- I've never used **forbidden words**, but by default students can access all maxima functions. Adding function names to this box prevents this, for instance if your question is to solve " $z^2 = i$ " then you may want to add "sqrt, ^" to this box. It seems unlikely they will know maxima functions, and might be a bit overkill.
- I've never used **allowed words**, but if you want students to access one of your variables (or functions) with a name longer than 2 characters then add it in the allowed words.
- **Forbid float** refuses to accept decimal answer when set as yes. I have always forbid float to prevent students giving " $0.33*a$ " instead of " $a/3$ " which is very difficult to give marks to when combined with algebra. However, if the answer is a number, I'd allow it, and then allow answers correct to 2 decimal places.
- **Require lowest terms** prevents submission if, for eg, $1+1$ is entered instead of 2. I always said no to this because I never trusted what STACK considers "lowest terms".
- **Check the type of response** prevents submission if, for e.g. a number is entered rather than list. I always selected no, as I don't trust what STACK considers a "type".
- **Student must verify** forces the student to wait until their parsed answer is displayed. I've always left this as yes.
- **Show the validation** changes what the validation box looks like, just leave it as default yes with variable list.
- **Extra options**, just leave it blank.

[Potential response tree: prt1](#)

This section is about how the marks should be given to the student.

- The **question value** dictates the weighting of one part of the question against another part.
- **Auto-simplify** will automatically convert a student's answer of " $1+1$ " to " 2 " before it has reached assessment tree.
- **Feedback variables** is a small maxima environment to process answers in more depth. It works in exactly the same way of question variables, except you can now refer to the

students answer(s) by the variable “ans1” and so on. Essentially, this section is used to calculate error carried forward answers.

- A **sketch of the response tree** will appear here to help see the bigger picture.
- At a node, you make checks of some kind. There are a huge amount of checks you can make, though in general, you’ll use this to test if the student answer matches with the teacher answer. So, for “**Answer test**”, you’ll generally use “AlgEquivalent” (algebraically equivalent, which is pretty general purpose, I rarely used any others except “NumAbs” when rounding was used), **SAns** you’ll use “ans1” and **TAns** use “ta1”. Nodes need not be limited to check student answers with teacher answer; you can check for equivalence between any two variables (for e.g. you might want to check if $\text{real_part}(\text{ans1}) = \text{real_part}(\text{ta1})$ and award half marks for this).
- If the node is true, you can add/subtract a **score** and then finish the process (via “**next: [stop]**”), or go to another node (via “**next: Node 2**”). Similarly, if the node is false, you can finish or go to another node.

Options

This section contains miscellaneous options.

- If **question level simplify** is yes then all inputs will be simplified before even getting to potential response trees.
- **Assume positive** I ignored, it’s not important. It’s about when a variable has used a Maxima command involving random variables which can produce random results, though your Maxima code really should be good enough to avoid this from ever being needed.
- **Multiplication sign** changes the way that multiplication is displayed in latex, e.g. is “a*b” displayed as “a\cdot b” or “a\times b” or “a\; b”. I almost always used “None” unless the question was entirely arithmetic, as dots and crosses made enormous and ugly equations where algebra was littered with dots.
- **Surd for square root** is about how to display “ \sqrt{x} ”
- **Meaning and display of $\sqrt{-1}$** is about how to type in and display $\sqrt{-1}$. Could be j for engineers or if using “i,j,k” vector notation then set this to “isymb” so that it does not confuse the two.
- **Inverse trigonometric functions** is about how to display inverse trig functions
- **Default shape of matrix parentheses** is about how to display matrices
- The **hints** is where official Moodle hints are meant to be placed. If these are used then (if “adaptive marking” is selected) each time a student submits an incorrect solution, the student is given another hint. Once the final hint is given, students are given a final chance to answer the question before moving on. Hint boxes will process have the same environment as question text, specific feedback, general feedback.

Others

- **Tags** has no purpose that I’m aware of
- **Created/last saved** shows which users have made and edited the question most recently
- The **fix dollars** changes $\$x\$$ and $\$x\$\$$ to $\$(x\)$ and $\$(x\)$ upon save if ticked. It seems like STACK didn’t used to allow dollars for LaTeX, but I’ve never encountered this as an issue

except in an older version of STACK, so it seems like this has been updated. Better safe than sorry, tick this when you're finished.

Walkthrough of a Basic Question

This section talks through the process of writing a STACK question in full detail with the intention that it will give sufficient groundwork for you to go off and develop your proficiency independently. The demo questions should provide a demonstration for most of the STACK techniques that you may wish to use and documentation referred to in the glossary section should provide more specific direction. Questions with more advanced mathematical content may require you to delve further into the Maxima functions which are referenced on the Maxima Syntax page.

It is recommended that you write your questions in waves, as you gain confidence you will undoubtedly merge some steps together but there are always advantages to writing in waves. Your first wave should lay out the skeleton for the question and then you should just add more layers of complexity each time. The main reason for this is that you are not able to save a partially complete question, it does not have to be polished but if you try to do it all in one go then you will get bogged down in one section and eventually realise that you can't save your last half hour's work. Another reason to work in waves is that it gives you the opportunity to test the question at various stages and fix problems as they appear in a relatively logical order.

The rest of this section will now focus on designing a question which tests a student's ability to add complex numbers. You could simply follow the tutorial exactly, in which case the text could be copied and pasted into STACK, or follow through and alter it as a question of multiplying complex numbers. I have included mentioning several common issues which you will probably come across in writing your own questions. However, this is a very simple example so some will not actually appear to be an issue in this case and some will only be apparent in the multiplication variant, either way, you will have to take it on faith that these are common problems/ solutions which will be useful in the future. In the text below, bold text indicates the section of the form you should be looking at and italics denotes text which could be copied and pasted into the STACK form, the rest is instruction or simply thinking out loud.

Preliminary steps:

- Log in to MoodleX and go to the specific module page
- Click on "Turn editing on" in the top right
- Click on "Question bank" on the left
- Click on "Create a new question" and select "STACK"

Authoring stage 1: Basic outline of the question

- Fill in **Question name**, this is only seen from the administrator's position. This shall be called *Basic addition of complex numbers*
- **Question variables:**
 - I want to check they can add and subtract complex numbers. They should be able to calculate $x+y$ and solve $z+y = x$ for given x and y
 - Hence, I want 4 complex numbers, call them x_1, x_2, y_1, y_2 . They should have random integer coefficients between -10 and 10. I will call my model answers ta_1 and ta_2 (teacher answers 1 and 2).
 - I would type this as:

```
x1: rand(21)-10 + (rand(21)-10)*i; x2: rand(21)-10 + (rand(21)-10)*i;
ta1: x1+x2;
y1: rand(21)-10 + (rand(21)-10)*i; y2: rand(21)-10 + (rand(21)-10)*i;
ta2:y1;
```

▪ **Question text:**

- I want the question to read:

“Calculate $x_1+x_2 = \underline{\hspace{1cm}}$

Solve $z + y_2 = y_1+y_2$. $z = \underline{\hspace{1cm}}$ ”

Where x_1 , x_2 , y_1 and y_1+y_2 are filled in by value.

- I would type this as:

```
Calculate \((@x1@) + (@x2@) = \) [[input:ans1]][[validation:ans1]][[feedback:prt1]]
Solve \((z + (@y2@) = @y1+y2@, z = \) [[input:ans2]][[validation:ans2]][[feedback:prt2]]
```

- The $\backslash(...\backslash)$ put you into inline latex environment
- $@...@$ creates a basic maxima environment, it can access variables from “question variables” and displays an evaluation of the statement
- You will gradually realise how you can use and position the “validation” and “feedback” sections but for now this is my recommended default
- **Default mark:** The first part is out of 1 and the second part is out of 1 so this is set to $1+1 = 2$
- **Specific feedback:** delete “[[feedback:prt1]]” from here because you put it in the question body instead
- **Penalty:** I wouldn’t punish people for trying so I set this to 0
- **Question note:** This must be enough detail for you to be able to reconstruct the exact question, basically an abbreviation of the question text. I would type:

```
Calc \((@x1@ + @x2@) and solve \((z + @y2@ = @y1+y2@)
```

Click on “**Verify the question text and update the form**”

- This will check you haven’t made silly mistakes (like missing steps e or g)
- This will also adapt the following sections now that it knows you want 2 input boxes and 2 response trees
- **Input: ans1/ans2:** These will be nearly identical so I will go through them together.
 - Both are numerical inputs so leave it saying “Algebraic input”, this is by far the most common input type
 - Model answer is $ta1/ta2$
 - The rest can be left for now
- **Potential response tree: prt1/prt2:** These will be identical too
 - Go down to Node 1
 - SAns is (student answer) $ans1/ans2$
 - TAns is (teacher answer) $ta1/ta2$
- Click on “save changes and continue editing” or “save changes”. You now have the skeleton of your question which will ask your question, check your answers and give you a mark for each correct answer

Preview stage 1.1:

- Click on “preview”, either next to the “save changes and continue editing” button or the magnifying glass next to the question name in the question bank
- First problem: your first question reads “Calculate $(0+0i) + (1+0i) =$ ”
 - This is a perfectly random question but not useful for assessments
 - This can be fixed by going back to your “Question variables” and replacing variable definitions with:

```
x1: rand_with_prohib(-10,10,[0]) + rand_with_prohib(-10,10,[0])*i;  
x2: rand_with_prohib(-10,10,[0]) + rand_with_prohib(-10,10,[0])*i;  
y1: rand_with_prohib(-10,10,[0]) + rand_with_prohib(-10,10,[0])*i;  
y2: rand_with_prohib(-10,10,[0]) + rand_with_prohib(-10,10,[0])*i;
```

This is a generally a much more useful and versatile way of generating random numbers

- Look [here](#) for more details on this topic
- Second problem: the variables have been printed “i+1” rather than “1+i” as you’d prefer
 - This can be fixed by going back to your “Question variables” and adding the line:
powerdisp:true;
 - This issue does occur occasionally, using a combination of this and the function [ratvars](#) should allow you to display equations as you want
- Third problem: your second question reads “Solve $z + (2-5i) = (3+2i)+(2-5i)$ ”
 - A completely correct evaluation of “@y1+y2@” but not in its simplest form
 - This can be fixed by going back to your “Question text” and replacing “@y1+y2@” with:

`@ev(y1+y2,expand,simp)@`
 - This is a very common problem, STACK does not use the same automatic evaluation simplification techniques that maxima uses so a lot of the time you will have to force it to simplify your expression as above
 - The “ev” function is almost always the function to turn to with this sort of problem, for more involved cases look [here](#) for more options

Preview stage 1.2:

- Once you fix the previous minor issues you actually try to type in some inputs
- For debugging it is recommended that you change “Attempt options” → “How questions behave” to “Adaptive mode (no penalties)” because it will let you try multiple inputs without refreshing the whole page which saves time
- The question reads “Calculate $(8+2i)+(-1+2i)$, Solve $z+(1-3i) = 2$ ”
- Press “fill in correct responses” to check that the model answers are actually being calculated correctly, you haven’t made a mistake somewhere in “question variables”
- Press “check” to confirm you do in fact get 100% for the model answers.
- Now retry with a couple of random, not right solutions to check that you get 0,1 and 2 marks when you should
- Now try just typing in “ $(8+2i)+(-1+2i)$ ” and “ $2-(1-3i)$ ” and press check. Unsurprisingly maxima is quite happy to compute such simple things for you and confirms that they are the correct answers

- Also note that if you'd just had a,b,c,d as your variables then it would have also happily accepted "a+b" and "c" as your answers. In general, all variable (or function) names with fewer than 3 characters can be accessed in the answer boxes.
- One solution can be found by going to "potential response tree: prt1" and performing the following steps:
 - Set "auto-simplify" to false (this means it won't simplify the expression before it even reaches you)
 - Change "AlgEquiv" to "EqualComAss" (this means that, if your model answer is a+bi then it will only accept: a+bi, bi+a, ib+a and a+ib. For a more detailed explanation see [here](#))
 - Click "add another node"
 - On "Node 1 when false" select "Next" to be "Node 2"
 - Change Node 2 to read as before: "AlgEquiv" → "ans1/2" → "ta1/2"
 - Add feedback to "Node 2 when true":
Your answer is correct but is not in its most simple form
 - Change the score in "Node 2 when true" to 0.5 to award half marks for the partially correct answer

Authoring stage 2: Adding some common mistakes feedback

- I propose that a common mistake for adding complex numbers is that someone might sum the real parts correctly and then forget to sum the complex parts
 - In "feedback variables" of "Potential response tree:prt1" define the following:

$$\text{mis1: is(realpart(ans1) = realpart(ta1)) and member(imagpart(ans1),[imagpart(x1),imagpart(x2)])};$$
 - Now add another node and ensure that "Node 2 when false" connects to Node 3
 - Make the test for Node 3: "AlgEquiv" → "mis1" → "true"
 - Write in "Node 3 true feedback":
You appear to have forgotten to sum the complex components

Preview stage 2:

- Everything should look right this time
- It is normally easier to "Fill in correct responses" first and then tweak them to give the desired "mistakes"
- Either way, just ensure that when you input your standard mistakes and click check then it gives you the intended feedback

Authoring stage 3: Add a hint and a model solution

- As a general hint, give people the general equations that they should be using. In the "specific feedback" box type:

Recall the relevant formulae:

$$\lfloor (a+bi) + (c+di) = (a+c) + (b+d)i \rfloor$$

$$\lfloor z + (a+bi) = (c+di) \text{ iff } z = (c+di) - (a+bi) \rfloor$$

For a fully worked solution, in the "general feedback" box type:

Model Solution:

$$\begin{aligned} (x_1) + (x_2) &= (\operatorname{Re}(x_1) + \operatorname{Re}(x_2)) + \\ &+ (\operatorname{Im}(x_1) + \operatorname{Im}(x_2))i \quad \&= \operatorname{Re}(x_1) + \operatorname{Re}(x_2) + \\ &+ (\operatorname{Im}(x_1) + \operatorname{Im}(x_2))i \end{aligned}$$

$$\begin{aligned} z + y_2 &= y_1 + y_2 \quad \& \text{iff } z = y_1 + y_2 - (y_2) \quad \& \text{implies } z = \\ &= (\operatorname{Re}(y_1 + y_2) - \operatorname{Re}(y_2)) + (\operatorname{Im}(y_1 + y_2) - \operatorname{Im}(y_2))i \quad \&= \\ &= \operatorname{Re}(y_1) + \operatorname{Im}(y_1)i \end{aligned}$$

[Preview stage 3:](#)

- The fastest way to test this part is just to press “fill in correct responses” then “submit and finish”
- Hopefully, the question should now be complete.

Walkthrough of an Advanced Question

Hopefully you will have completed the basic question and have a reasonable understanding of how STACK questions work and should be written.

In this walkthrough, we will introduce some more complex concepts using randomisation and novel techniques in Maxima. Again, we shall write the question in waves, as this is good habit and you should continue to write your question like this (to a certain extent), even as you become more advanced and grow comfortable with STACK.

The question will be about finding the gradient of a function, ϕ , and finding the i -component after evaluating it at a point. We shall create the skeleton of the question, make some minor adjustments, and then add our more complex ideas. The tutorial should be followed exactly, though some of the more complex ideas can be introduced at any point.

Preliminary steps:

Log in to MoodleX and go to the specific module page. Click on “Turn editing on” in the top right. Click on “Question bank” on the left. Click on “Create a new question” and select “STACK”.

Stage 1: Creating the skeleton of the question

Fill in **Question name**, this is only seen from the administrator’s position. This shall be called *Finding the Gradient*

Question variables:

What kind of variables will I want in here? The bare minimum we will need will be the function ϕ , the gradient of ϕ , the point we’re evaluating at, and that point evaluated at the gradient of ϕ . Let’s keep it simple for now and pick a fixed function of ϕ and a fixed point, and work out the solutions for those:

```
phi: x^2+y^2+z^2;  
gradphi: 2*x*i+2*y*j+2*z*k;  
point: [1,2,3];  
ta1: gradphi;  
ta2: 2;
```

Question text:

I highly recommend that you start writing your questions in a LaTeX environment instead of a writing in the awful STACK environment. It’s so much easier to detect errors in something like TexStudio, and it can act as an extremely handy back up when things go wrong (which can certainly happen, see debugging for details). Whilst it isn’t perfect to work in a strictly LaTeX environment, and any maxima you write will look odd in a LaTeX pdf, it’s surely better than the STACK environment which simply acts like a notepad. Go and write your question in TexStudio. Then, switch to HTML view on the question text, paste the LaTeX and then close the HTML view. You must paste into an HTML view instead of normally, else STACK will not display it in the question, as STACK imposes weird formatting to it if it is pasted in non-HTML settings. Once pasted and switching back to normal view, it will come out like this:

For a function $\phi = \phi(x, y, z)$, find the gradient of ϕ denoted by $\nabla \phi$. $\nabla \phi = \begin{pmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial z} \end{pmatrix}$ Now, find the value of the i -component in $\nabla \phi$ at (x_1, y_1, z_1) . i -component of $\nabla \phi (x_1, y_1, z_1) = \frac{\partial \phi}{\partial x_i}$

It's pretty ugly compared to TexStudio, but it works. The only problem with this is that the formatting will be wrong. Hence, we have to edit it such that the line spacing in non-LaTeX parts is nice. So, change it to this:

For a function $\phi = \phi(x, y, z)$, find the gradient of ϕ denoted by $\nabla \phi$.

$\nabla \phi = \begin{pmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial z} \end{pmatrix}$

Now, find the value of the i -component in $\nabla \phi$ at (x_1, y_1, z_1) .

i -component of $\nabla \phi (x_1, y_1, z_1) = \frac{\partial \phi}{\partial x_i}$

Default mark: The first part is out of 1 and the second part is out of 1 so this is set to $1+1 = 2$

Specific feedback: delete "[[feedback:p1]]" from here because you put it in the question body instead

Penalty: I wouldn't punish people for trying so I set this to 0

Question note: This must be enough detail for you to be able to reconstruct the exact question, basically an abbreviation of the question text. I would type:

Find the gradient of ϕ and the i -component of the gradient evaluated at (x_1, y_1, z_1) .

Click on "Verify the question text and update the form". This will adapt the following sections now that it knows you want 2 input boxes and 2 response trees.

Input: ans1/ans2: These will be nearly identical so I will go through them together. Both are numerical inputs so leave it saying "Algebraic input", this is by far the most common input type. Model answer is ta_1 and ta_2 for ans1 and ans2 respectively. The rest can be left for now

Potential response tree: prt1/prt2: These will be identical too. Go down to Node 1 on prt1, SAns is student answer, so enter ans_1 , TAns is teacher answer so enter ta_1 . For prt2, go to Node 1, SAns is student answer, so enter ans_2 , TAns is teacher answer so enter ta_2 .

Click on "save changes and continue editing" or "save changes". You now have the skeleton of your question which will ask your question, check your answers and give you a mark for each correct answer

[Preview stage 1:](#)

Click on "preview", either next to the "save changes and continue editing" button or the magnifying glass next to the question name in the question bank

- First problem: ϕ appears as $z^2+y^2+x^2$

- I've never quite been able to work out exactly how Maxima orders its variables in functions by default, but I know that the best way this can be solved is by typing "*powerdisp:true;*" at the start of your question variables. In fact, I'd recommend you do this for every single question you ever make. This means that functions are ordered in alphabetical order of variables. It isn't perfect, but it's the best there is. If you want to try better methods, look through the Maxima documentation (it seems like the function "*ratvars*" should work, but STACK doesn't process all Maxima commands), but I've had trouble finding anything that works. On this subject, I'd recommend you use "*factor*" as much as you can with functions. I've found it to be the most sensible and simplest way of writing functions compared to all other Maxima commands.
- Second problem: "*i*" isn't understood as a variable, but a complex number;
 - Always a problem for vector questions, but easily solved. In the "Options", change "Meaning and Disp of sqrt(-1)" to "*symi*". Now it will understand *i* as a variable rather than complex number.

Stage 2: Making the question more complex

- I would like to make ϕ a random function instead of a $x^2+y^2+z^2$
 - Depending on how you want it to be randomly created, this can be completed in many ways. Creating a list of functions and selecting random elements of this list can be easily implemented, along with random coefficients. However, one method I found very powerful in many respects was the follow method of selecting from lists without repeats. This technique will be powerful in many regards, as it is the only way of using non-repetition from lists. We shall create ϕ using random, non-repeated functions from a list.
 - This is the code we will need:


```
FuncList:[sin,cos,exp,log];
Func:[rand(length(FuncList))+1,0,0];
Func[2]:rand_with_prohib(1,length(FuncList),[Func[1]]);
Func[3]:rand_with_prohib(1,length(FuncList),[Func[1],Func[2]]);
phi: FuncList[Func[1]](x) + FuncList[Func[2]](y) + FuncList[Func[3]](z);
```
 - How does this work? We create a function list, *FuncList*. It may be tempting to try and code something along this lines of "*rand_with_prohib(Func,[Func[1]])*" but the function *rand_with_prohib* **does not work like that**, it only works with numbers, and no such function like it exists. Instead, we have to set up this list, *Func*, of non-repeated numbers in the range of the length of *FuncList*, which we will use as indices to draw functions from our function list. Moreover, we have to specify the arguments of *Func* 1-by-1, as we cannot exclude variables from a variable before it has properly been defined.
 - For more examples of this technique, go to 2016/17 → PX276 → Test 3 and then any of the True/False questions.
- I need to change the gradient of ϕ .
 - Correct, we'll need a maxima function. This is an excellent opportunity for me to introduce the concept of functions, so we shall to do this by defining a gradient function.

There's two different ways to define functions in maxima: symbolically and non-symbolically. The way we defined phi is symbolic, and I recommend doing this almost every time, as it can be differentiated, evaluated (by using "*subst*"), etc. The other way is as follows:

```
grad(f) := diff(f,x)*i + diff(f,y)*j + diff(f,z)*k;
```

- This is useful if you never intend on displaying it and only ever evaluating things with it, because it's so much quicker and cleaner than using "*subst*". Hence, we let

```
gradphi: grad(phi);
```

- I want a randomised point, and I want it to be suitable for my function phi.

- This is the code we will need:

```
NiceCoords:[[0,pi/3,pi/2, pi],[0,pi/6,5*pi/6], [0,log(2),log(3)], [1,2,3]];
point:[rand(NiceCoords[Func[1]]), rand(NiceCoords[Func[2]]),
rand(NiceCoords[Func[3]])];
```

- How does this work? I've created a list of list of nice co-ordinates. The first index of *NiceCoords* has a list nice points for the differential of the function which is the first index *FuncList*. Even though I don't explicitly know phi, I know what the function with x as an argument will be by the saved random number *Func[1]*. So, the x-value of *point* is a random nice number for the matching function. Similar for y and z.

- I don't want them to give me the i-coefficient, but a random one of i,j,k.

- Nice idea, this will stop students cheating, especially when you're struggling to randomise the question. This will introduce the idea of case scenarios that you will frequently encounter and deploy. This will work by saving values of different cases in lists, where each case is contained in the same index in all the lists.
- The code to do this is given by:

```
gradatpoint: subst([x=point[1],y=point[2],z=point[3]],gradphi);
case: rand(3)+1;
ijk: [i,j,k];
solutions2:
[coeff(gradatpoint,i),coeff(gradatpoint,j),coeff(gradatpoint,k)];
dir: ijk[case];
ta2: solutions2[case];
```

- Firstly we calculate the gradient at the point using *subst*. We note there are three different cases, so we generate a case-defining random number of either 1, 2 or 3.
- We save values in lists (*ijk* and *solutions2*) for each of the cases, where the index matches with the case for every list we have.
- Then, we save the final variables we will use (*dir* and *ta2*) by using these lists and calling the index of the case-defining number.
 - (If you're quick-witted, you might have noticed we could have taken a shortcut by letting "*ta2: coeff(gradatpoint,ijk)*", and this would work, however I was trying to demonstrate the concept of using lists and indices as ways of saving cases. There will be plenty of times when shortcuts like this can't be used.)

- Remember to change any reference to “i” direction made before in the question to “@dir@”. You might want to make that bold font too, as it is a vector.
- For more advanced examples of this technique, go to 2016/17 → PX275 → Term 1 Test 2 → Calculate curl and line integral. This question uses an enormous amount of complex case scenarios.
- I want a general solution.
 - Do this in the “General Feedback”. As like in the question text, I can’t advise strongly enough that you write this in TexStudio and then copy and paste it into HTML view. This section will be almost entirely written in LaTeX, so why on earth wouldn’t you write it in a LaTeX environment. Moodle is awful for typing LaTeX as it won’t pick up errors until you try to save it. There’s also some pretty nasty errors that can occur making you lose work (see the debugging), so having a back-up is highly advised.
 - After you copy and paste into HTML view, the formatting is generally ok, but sometimes it will have little mistakes like not processing line breaks, and this can be easily changed in the non-HTML view by typing enter. You will get used to having to constantly render tiny formatting mistakes.
 - Remember to refer to variables and maxima commands as much as you want. In fact, I’d advise you do it as much as you can, it’s good practice and you’ll realise the capabilities the integration between LaTeX and Maxima has to offer. You may find yourself wanting to work in LaTeX as much as possible and then just using the Maxima variables whenever needed, but this is bad habit! You will encounter stupid issues like having “- 1*x” instead of “x” appear, which comes from typing stuff like “-@coeff@x” instead of “@-coeff*x@”. Learning the appropriate amount of Maxima commands integrated into LaTeX will come with experience.
 - Here is an example solution we shall use (in HTML view):

```
<h4>Model Solution</h4>We calculate \begin{align*} \nabla\phi &= \frac{\partial \phi}{\partial x}\mathbf{i} + \frac{\partial \phi}{\partial y}\mathbf{j} + \frac{\partial \phi}{\partial z}\mathbf{k} \quad \&= @diff(phi,x)@ \, \mathbf{i} + @diff(phi,y)@ \, \mathbf{j} + @diff(phi,z)@ \, \mathbf{k} \end{align*} So, evaluating \(\phi\) at \((x,y,z) = \left(@point[1]@, @point[2]@, @point[3]@ \right)\) gives us \begin{align*} \nabla \phi \left(@point[1]@, @point[2]@, @point[3]@ \right) &= @subst([x=@point[1],y=@point[2],z=@point[3]], diff(phi,x))@ \, \mathbf{i} + @subst([x=@point[1],y=@point[2],z=@point[3]], diff(phi,y))@ \, \mathbf{j} + @subst([x=@point[1],y=@point[2],z=@point[3]], diff(phi,z))@ \, \mathbf{k} \end{align*} Hence, the $\mathbf{@dir@}$ coefficient is @ta2@.
```

- I want to allow rounded answers to two decimal places for the second part.
 - Probably best to specify this in the question text first. **Always** tell the students what they can and can’t do, never leave anything unsaid.
 - Go and change “forbid float” to “no” in ans2.
 - Then, in the potential response tree 2, change “AlgEquiv” to “NumAbs”. These are the only two settings I ever used for nodes.
 - Change the Test options to 0.005001. “NumAbs” works such that the node will be true if |SAns – TAns| < 0.005001. We needed that stupid 1 in there because if we didn’t, it

wouldn't accept 2.345 rounded to 2.35 as true. Hence, this setting is equivalent to being correct to two decimal places.

- Should I give any syntax hints?
 - Yes, do it for ans1 because we're using i,j,k vectors. We've had problems before of people not knowing whether to do i,j,k or matrices or whatever. Try as hard as you can to make them enter something in the right way, or at least tell them in the question what you require of them, so they can't complain if they aren't given the marks.
 - Enter "(?)*i + (?)*j + (?)*k" in the syntax hint for ans1.
- I want error carried forward marks if they get part (a) wrong.
 - Go to the feedback variables in potential response tree part 2. We will want to calculate the 'error carried forward correct answer' and save them as variables
 - Define the feedback variables:

```
ecfgradatpoint: subst([x=point[1],y=point[2],z=point[3]],ans1);  
  
ecfsolutions2:  
[coeff(ecfgradatpoint,i),coeff(ecfgradatpoint,j),coeff(ecfgradatpoint,k)];  
  
ecf2: ecfsolutions2[case];
```
 - Essentially, we're defining variables we used in the initial setup here, but using the wrong answer of "ans1" carried through.
 - Now we want to add another node on the potential response tree.
 - Click "add another node".
 - Change the "next" on "node 1 when false" to "node 2".
 - In node 2, do the exact same settings of NumAbs and 0.005001 (they can have a rounded ecf ans too!), but refer to ecf2 instead of ta2 as the check. Remember to add 1 mark on if this is true.
- I don't like all the dots for multiplications, I want to change them.
 - Me neither. In "options", change "multiplication sign" to "none".
- I'm worried they won't know how to enter functions
 - I worry this too. Insist that the module organiser puts some STACK documentation about how to enter equations somewhere on the module page. Also, I always tried to be as explicit as possible in the question text and give them everything they'll need. For example, I'd always say: Note that (e^x) can be typed as either `e^(x)` or `exp(x)` in the answer box.
- It looks a bit messy, I want to tidy it up
 - Don't worry about the LaTeX looking messy, that's almost completely unavoidable, it won't look messy when displayed. The only thing you can (and should) do is put some commentary in the question variables, explaining the purpose of each part of the code. Comments can be added using forward slashes and asterisks like `/* this */`. A well organised set of question variables will be so helpful to yourself and anyone else trying to modify the question.

[Preview stage 2:](#)

Everything should look right this time. It is normally easier to "Fill in correct responses" first and then tweak them to give the desired "mistakes". Either way, just ensure that when you input your

standard mistakes and click check then it gives you the intended feedback. Hopefully, the question should now be complete.


Hints and Debugging

Hints

- Complex numbers will not automatically expand, if you want to have $(1+3i)(2-i)$ displayed as $5+5i$ then you have to type `"ev(____,expand,simp)"` around it
- By default, trigonometric expressions doesn't simplify to what we would consider its simplest form. Use `"trigsimp()"` as much as possible to simplify expressions. Remember, it won't be perfect, so be sure to test and see what comes out
- You will see this remark in a couple of other places but it's worth pointing out here that you can't use line breaks to format your variables, it will interpret each line as a different statement
- To format your latex nicely such as for align environments then type it in within the HTML display ("Show more buttons" → "HTML"). This view does not care about white space such as new lines or tabs. I can't advise strongly enough that you write this in TexStudio and then copy and paste your LaTeX into HTML view. Moodle is awful for typing LaTeX as it won't pick up errors until you try to save it, and the error messages will be vague. There's a horrible error that can make you lose work too (see debugging), so it's highly advised you keep backups whilst you work. After you copy and paste into HTML view, the formatting is generally ok, but sometimes it will have little mistakes like not processing line breaks, and this can be easily changed in the non-HTML view by typing enter. You will get used to having to constantly render tiny formatting mistakes.
- STACK likes fractions a lot more than decimals, if you want to allow decimals or standard form then you will have to set "Input" → "Forbid float" to false.
- The default is to give a penalty of 0.1 each time a student gets the answer wrong (when they're given multiple attempts at a question). i.e. by default, each time a student tries again they lose 10% of their mark and after 10 attempts they will get 0 each time. It is of course down to the individual lecturers what to do with this, but in my case I had to remember to set this to 0 each time.
- If you are writing vectors in $"i,j,k"$ notation then this is fine, maxima will view this as a polynomial in i,j,k , but it will also by default think that i is the complex number which could lead to some odd properties. If you change "Options" → "Meaning and display of $\sqrt{-1}$ " to "isymb" then this won't be a problem.
- For display reasons, you will mostly like want to set "Options" → "Multiplication sign" to "none" to make things look a little smoother.
- In your question variables, you may want to add `"powerdisp:true;"`, which will write polynomials in a more standard form.
 - The function variable ordering is quite odd, but this is surely the best option I've found. If you really want to find some Maxima command to get something exactly how you like, feel free to search their documentation. Personally, I found it very confusing and nothing seemed to work. "ratvars" might be a start, but don't be surprised if nothing makes a difference.
 - One small trick I found was using `"."` instead of `"*"` for multiplication in Maxima. The fullstop means "non-commutative multiplication", so it forces monomials to stay in a certain order. It helps in some cases, but now you're stuck with a dot you can't get rid of like you could with the multiplication one.

- You may find yourself wanting to work in LaTeX as much as possible and then just using the Maxima variables whenever needed, but this is bad habit! You will encounter stupid issues like having “- 1*x” instead of “x” appear, which comes from typing stuff like “-@coeff*x” instead of “@-coeff*x@”. Try to using Maxima commands as much as possible. Learning the appropriate amount of Maxima commands integrated into LaTeX will come with experience.
- Don’t forget to use @’s around your Maxima variables and commands. It’s hard to spot in any environment, be it TexStudio or Moodle, but it’ll be obvious in the Moodle previews.
- If you’re struggling to get used to Maxima, there is Maxima documentation, though it’s long and complicated. The STACK documentation is more likely to help you than that, unless you’re searching for something very specific. In fact, your best bet is probably reading some previous questions and seeing how they work.
- Graphs are surprisingly easy to plot. It will even plot piecewise functions if you wish (use some if statements to define them, along with lambda). Define your function f symbolically, for example, f: lambda([x], if (x>=-1 and x<=1) then 1 else 0);. Note that Moodle sometimes struggles to process two commands in an if statement check, however the lambda seemed to solve this. Then, in the question text, simply type @plot(f(x),[x,-2,2],[y,-2,2])@. There is some good STACK documentation about plots anyhow.
- Check your maths properly for every question, so that you aren’t ignoring cases where things can go horribly wrong (negative logs, integrating non converging integrals, etc.)

Debugging:

- When I save the question, sometimes the LaTeX no longer displays as TeX code, but as pretty, displayed math.
 - This is a **horrible** problem that really needs fixing by the IT department. Please report this to them. It occurs around 15% of the time when you save your question, but here’s how to deal with it:
 - Refresh the page. This will solve the problem about half the time. It’ll either put it back in tex code, or make no difference, or give blank images (it doesn’t even display the pretty math properly)
 - If that doesn’t work, there is a “back” button among the buttons (such as bold text, HTML view, etc.) which goes back to previously saved versions in the question text/feedback/etc. It looks like this: . Click it and try to restore a previous version where it is in tex code like you want.
 - If it this button cannot be clicked, then you’re a little bit screwed. If you’ve taken my advice of working in TexStudio, it’s annoying, but just a copy and paste job, with some formatting rendering. Otherwise, I’m afraid to say, you might have lost some work here. Maybe close the whole page and come back to it?
- I put a picture in a question. Now I’ve saved it, and it won’t display in the question or the previews.
 - This should be reported to the IT department too. Sadly, I can’t find a single fix for this. All you can do is import the image again. It might be for the best if you import the image at the end when you’re done, instead of keep having to do it whilst you’re working.
- LaTeX isn’t being displayed in my question preview, but shows as its TeX code:
 - Check that you’ve closed all your brackets and environments. You most likely have a mistake in your code. If you’re not working in TexStudio already, write you LaTeX in that

first and then copy and paste it over into the HTML view. You'll be able to see if it's the code that's the problem that way. If it works in that, it should work in your display.

- If it's working in TexStudio but not displaying in the Moodle preview, then if you've copied and pasted bits then in the non-HTML, it may have been wrapped in "`<span...> ____ `" elements. This will often break the latex execution and be completely invisible from the front view. To get rid of this, highlight the section of text with errors in and click "Clear formatting". To prevent this problem in the long term, it is recommended to go into the HTML view to paste or write complicated latex text.
- If none of these work, perhaps you're running a complex LaTeX command which Moodle isn't capable of running. Moodle doesn't have all the capabilities to run LaTeX yet, and you won't be able to run that complex part of your code, sorry.
- Are you trying to put any validation boxes or answer boxes inside LaTeX code? You can't do that, sorry to say.
- "CAS failed to return any evaluated expressions. Please check your connection with the CAS"
 - You probably have some invalid Maxima. STACK gives pretty vague errors responses, so good luck finding the error. I'd recommend you build your questions up bit by bit and save it frequently to help spot errors.
 - You might be doing something that STACK thinks it can do but actually can't. STACK doesn't have the full capabilities of Maxima, so more complex commands it cannot process
 - It will complain if you have a variable (or function) name with numerical characters in the middle, such as if I called a variables "a1a".
- "You seem to be missing * characters. Perhaps you meant to type..."
 - STACK is a lot more sensitive to formatting than maxima is. Multiplications always need a * and you cannot write 2a for example, it needs to be 2*a.
 - Often would sometimes like white space between brackets and characters, even if it's valid maxima
- An error displays about evaluating something when I try to preview the question
 - You most likely have some invalid mathematics in here. Something like trying to evaluate a negative log, or integrating a non-finite integrand. Check your maths thoroughly, this can happen easily if you're messing around with randomised functions and such, you've probably ended up with a weird case you need to exclude.

Basic Maxima Syntax

Some of the challenge in setting questions in STACK is to find the right Maxima syntax to calculate your answers and any model answers. Most of Maxima functions are preloaded for use but there may be some that you wish to add to your own “library” and copy and paste the function declaration in each time you wish to use it. Below are some of the most common functions and an introduction to the basic syntax of maxima data types. The official contents page for Maxima can be found [here](#) and the index of all functions, [here](#).

Linguistic definition	Maxima syntax	Description
Set constant, c, to 5	c: 5;	Basic method for setting constant variables.
Define symbolic function $f(x) = \cos(x)$	f: cos(x+y);	This form of definition can be dealt with symbolically, e.g. differentiated and integrated exactly. To evaluate f at a point, e.g. (1,2), use the syntax subst([x=1,y=2],f). For one variable functions, i.e. $f = \cos(x)$, use subst(1,x,f).
Define function for evaluation $f(x) = \cos(x)$	f(x) := cos(x);	This form of definition cannot be dealt with symbolically however to evaluate f at a point, e.g. 0, use the syntax f(0).
Make an ordered set of: a,c,a,b	s:[a,c,a,b];	This is parsed as the List object s=[a,c,a,b], exactly as typed. The elements are accessed as s[1],s[2],... and one can define a multi-dimensional array as a list of lists, not necessarily of equal lengths. For more functions concerning lists look here .
Make an unordered set of: a,c,a,b	s:{a,c,a,b};	This is parsed as the Set object s={a,b,c}, it has been somehow sorted and repeated elements removed. You can access the elements as first(s),second(s),...,last(s) but the more natural method is to perform a “for each x in s...” iteration. For more functions concerning sets look here .
Convert list to set	setify(s);	Converts [a,c,a,b] to {a,b,c}.
Convert set to list	listify(s);	Converts {a,b,c} to [a,b,c].
Create 2x3 matrix	M:matrix([1,2,3],[4,5,6]);	This initiates a matrix where the values are accessed to give $M[i][j] = 3*i + j - 3$; Also remember that if you have an nx1 vector then you still have to access values as M[i][1] to return a value.
Calculate $d^2f/dxdy$	diff(f,x,1,y,1);	This same structure of diff can be extended indefinitely to calculate any degree of differential.
Integrate f dx	integrate(f,x);	To make this a definite integral, between a and b, you add the extra variables: integrate(f,x,a,b);.

This sums up the most basic Maxima syntax. As STACK does not allow full maxima functionality, there are also some tricks which you have to use in STACK which look a bit weird:

Linguistic definition	STACK-Maxima
For $k=1,2,\dots,n$, $L[k] = e^{(i*k/n)}$ end	L:makelist(e^(i*k/n),k,1,n);
T = number of terms such that $\text{Ans}[k]=L[k]$	T: length(sublist(L,lambda([k],is(Ans[k]=L[k]))));
Find k such that $L[k] = 1$	k:makelist(k,k,1,n); k:sublist(k, lambda([j], is(L[j]=1))); k:first(k);

Library of functions to be copied and pasted as needed:

Maxima	English
$\text{Grad}(f) := \text{diff}(f,x)*i + \text{diff}(f,y)*j + \text{diff}(f,z)*k;$	Calculates the operation ∇f
$\text{Dot}(u,v) := \text{coeff}(u,i)*\text{coeff}(v,i) + \text{coeff}(u,j)*\text{coeff}(v,j) + \text{coeff}(u,k)*\text{coeff}(v,k);$	Calculates the operation $u \cdot v$
$\text{Cross}(u,v) := (\text{coeff}(u,j)*\text{coeff}(v,k)-\text{coeff}(u,k)*\text{coeff}(v,j))*i - (\text{coeff}(u,i)*\text{coeff}(v,k)-\text{coeff}(u,k)*\text{coeff}(v,i))*j + (\text{coeff}(u,i)*\text{coeff}(v,j)-\text{coeff}(u,j)*\text{coeff}(v,i))*k;$	Calculates the operation $u \times v$
$\text{Curl}(f) := (\text{diff}(\text{coeff}(f,k),y)-\text{diff}(\text{coeff}(f,j),z))*i + (\text{diff}(\text{coeff}(f,i),z)-\text{diff}(\text{coeff}(f,k),x))*j + (\text{diff}(\text{coeff}(f,j),x)-\text{diff}(\text{coeff}(f,i),y))*k;$	Calculates the operation $\nabla \times f$
$\text{Div}(f) := \text{diff}(\text{coeff}(f,i),x) + \text{diff}(\text{coeff}(f,j),y) + \text{diff}(\text{coeff}(f,k),z);$	Calculates the operation $\nabla \cdot f$
$\text{Grad2}(f) := \text{diff}(f,x,2) + \text{diff}(f,y,2) + \text{diff}(f,z,2);$	Calculates the operation $\nabla^2 f$
$\text{simpVec}(f) := i*\text{factor}(\text{coeff}(f,i)) + j*\text{factor}(\text{coeff}(f,j)) + k*\text{factor}(\text{coeff}(f,k))$	Simplifies vectors in the i,j,k notation
$\text{Mag}(u) := \text{sqrt}(\text{coeff}(u,i)^2 + \text{coeff}(u,j)^2 + \text{coeff}(u,k)^2);$	Calculates $ u $ for the vector u in i,j,k notation
$\text{DirDeriv}(f,u) := (\text{coeff}(u,i)*\text{diff}(f,x) + \text{coeff}(u,j)*\text{diff}(f,y) + \text{coeff}(u,k)*\text{diff}(f,z))/(\text{sqrt}(\text{coeff}(u,i)^2 + \text{coeff}(u,j)^2 + \text{coeff}(u,k)^2));$	Calculates $\nabla_u f$, the directional derivative of f in unit direction u .