Change a ULethQuestionBank Question Checklist

This checklist will guide you through the basic things that you will need to do in most cases if you are modifying a problem from the ULethQuestionBank.

- ❏ If you want to add parts to the question, first check to see if the question contains instructions for this task. If it does, they will be right under the keywords section near the top of the file. Follow these steps to add your new questions; they are all listed at the top of the file, but are also located in the sections they refer to if you scroll through the file. Many of them also have examples to give you an idea of how you could add more to the problem. If these steps are not present, or if you want to do other modifications to the question, follow the checklist below.
- ❏ Change the description, date, author, and keywords in the .pg file to match the question. This is important, because it makes it easier to search for your problems in the WeBWorK library.
- ❏ Add or modify MathObjects to get the values required for your problem. Where possible, try to give them useful names that will help you keep track of them. For example, if you're adding a new question to a problem that already has two questions, `$answer3` or `$answerC` are much more useful as names than `$newAnswer`.
- ❏ Change the problem text to accommodate the new values/questions that you've added to the problem. If you've changed the names of existing MathObjects, you'll have to modify their names in the problem text so they display properly. Type everything between the `BEGIN_TEXT` / `BEGIN_PGML` and `END_TEXT` / `END_PGML` tags to ensure that it is processed properly.
    - ❏ If using `BEGIN_PGML` / `END_PGML`
        - ❏ LaTeX is entered as `[`Your LaTeX here`]`. Note that the marks next to the square brackets are backticks: **NOT** apostrophes!
        - ❏ MathObjects are entered as `[$myMathObject]`; this includes if they are being used inline with some LaTeX.
        - ❏ Lists (e.g. lists of questions) should have an empty line between each question. This will place each on a new line in the problem. You can use uppercase/lowercase letters, roman numerals, or numbers to number the questions, but keep it consistent! If you want to put something on a new line, but keep it indented under the current question

          (e.g. This line of text),

          then put a line of empty space between it and the previous line (see above), then put four spaces in front of it (not pictured).
        - ❏ Code that needs to be evaluated within the text of the problem (e.g. indexing a multidimensional array) is entered as `[@ Some code @]*`;

this can also be entered within a LaTeX expression if you want the code's result to display in LaTeX. Where possible, though, you should run your code outside of the solution text, store the results in MathObjects, and then display these in the solution text.

- ❏ If using `BEGIN_TEXT` / `END_TEXT`
  - ❏ LaTeX is entered as `\(Your LaTeX here\)`. Note that the marks next to the parentheses are backslashes.
  - ❏ MathObjects are entered as `$myMathObject`; this includes if they are being used inline with some LaTeX. In other words, they do not need any special characters around them.
  - ❏ Lists (e.g. lists of questions) should have a `$BR` (line break marker) between each question. This will place each on a new line in the problem. You can use uppercase/lowercase letters, roman numerals, or numbers to number the questions, but keep it consistent!
  - ❏ Code that needs to be evaluated within the text of the problem (e.g. indexing a multidimensional array) is entered as `\{ Some code \}`; this can also be entered within a LaTeX expression if you want the code's result to display in LaTeX. Note that the marks next to the curly braces are backslashes. Where possible, you should run your code outside of the solution text, store the results in MathObjects, and then display these in the solution text.

- ❏ Add answer blanks as needed.
  - ❏ If using PGML, use `[_____]`, where each underscore represents one character of width for the answer blank.
  - ❏ If using PG, enter `ans_rule(int)`, where `int` is some integer that determines how many characters wide the answer blank is.
- ❏ Set `$showPartialCorrectAnswers`
  - ❏ `$showPartialCorrectAnswers = 1` if you want to give feedback for partially-correct answers.
  - ❏ `$showPartialCorrectAnswers = 0` if you don't want to give feedback for partially-correct answers.
- ❏ Add answer checkers as required. In general, this will just be `ANS($answer->cmp());`, where `$answer` is the name of whatever value you want to compare against the student's answer. If there are multiple answer blanks in the problem, make sure that the answer checkers are in the same order as the blanks in the problem. This ensures that each checker is checking the correct answer blank.
- ❏ Add your solution text. This is text that will be displayed to the students after the due date. Change the solution text to accommodate the new values/questions that you've added to the problem. If you've changed the names of existing MathObjects, you'll have to modify their names in the solution text so they display properly. Type everything between the `BEGIN_SOLUTION` / `BEGIN_PGML_SOLUTION` and `END_SOLUTION` / `END_PGML_SOLUTION` tags to ensure that it is processed properly.
  - ❏ If using `BEGIN_PGML_SOLUTION` / `END_PGML_SOLUTION`

- LaTeX is entered as `[`Your LaTeX here`]`. Note that the marks next to the square brackets are backticks: **NOT** apostrophes!
- MathObjects are entered as `[$myMathObject]`; this includes if they are being used inline with some LaTeX.
- Lists (e.g. lists of solution parts) should have an empty line between each point. This will place each on a new line in the solution. You can use uppercase/lowercase letters, roman numerals, or numbers to number the solution parts, but keep it consistent! If you want to put something on a new line, but keep it indented under the current solution part

  (e.g. This line of text),

  then put a line of empty space between it and the previous line (see above), then put four spaces in front of it (not pictured).
- Code that needs to be evaluated within the text of the solution (e.g. indexing a multidimensional array) is entered as `[@ Some code @]*`; this can also be entered within a LaTeX expression if you want the code's result to display in LaTeX. Where possible, though, you should run your code outside of the solution text, store the results in MathObjects, and then display these in the solution text.
- If using `BEGIN_SOLUTION` / `END_SOLUTION`
  - LaTeX is entered as `\(Your LaTeX here\)`. Note that the marks next to the parentheses are backslashes.
  - MathObjects are entered as `$myMathObject`; this includes if they are being used inline with some LaTeX. In other words, they do not need any special characters around them.
  - Lists (e.g. lists of solution parts) should have a `$BR` (line break marker) between each point. This will place each on a new line in the problem. You can use uppercase/lowercase letters, roman numerals, or numbers to number the solution parts, but keep it consistent!
  - Code that needs to be evaluated within the text of the solution (e.g. indexing a multidimensional array) is entered as `\{ Some code \}`; this can also be entered within a LaTeX expression if you want the code's result to display in LaTeX. Note that the marks next to the curly braces are backslashes. Where possible, you should run your code outside of the solution text, store the results in MathObjects, and then display these in the solution text.

- Check for common errors
  - Make sure that all opening brackets have matching closing brackets (and vice versa). There are a few exceptions to this rule (e.g. when writing intervals with one closed endpoint and one open endpoint), but it's still important to check that you haven't missed something critical.

- ❏ Check that you've used backticks ``` to enclose any LaTeX within your problem text (assuming it's located within a `BEGIN_PGML` / `END_PGML` block) instead of apostrophes ''. The backtick key is located below the Esc key on most keyboards.
- ❏ Check for semicolons! All lines of code should end with a semicolon. This does not include commented lines (lines that start with a `#` symbol), block lines (e.g. `BEGIN_PGML` / `END_PGML` lines), or the problem/solution text (assuming it's within a block like the `BEGIN_PGML` / `END_PGML` block).
- ❏ Check that all MathObjects and Perl variables you are using have been properly initialized. This means that if you use a value (say, a MathObject called `$answer` that contains the question's answer), there should be a line somewhere earlier where you initialize that value (e.g. `$answer = random(1,10,1);`). This also includes misspelled names; misspelling a name will sometimes cause the question to crash, or may produce unexpected results during computations. In other words, setup things before you use them, and check your spelling!