Hiding Answers

How to hide answers from the right-click menu.

The Problem, right-clicking

In order to make Xronos accessible (as in DRC accessibility) we use a well supported and widely used library to render the mathematics in a way that is compatible with a whole lot of different accessibility options. One of these features is the automatic generating/providing of the LaTeX code that generates mathematics on the page (interestingly, this is the primary way blind people do mathematics online, they learn and use LaTeX syntax). So, if the original mathemode code is written in LaTeX, then that code is exposed to the student (via a right-click menu in some cases) as-is.

Unfortunately, this applies to *everything* in mathemode, which includes the `\answer` command (which must be in mathemode). Moreover, since the answer command has, as it's argument, the actual *answer* that Xronos is looking for, that means the content of the answer command (a.k.a. the actual answer for the problem) is directly available to students. By way of example:

Exercise 1 *Right click the following answer box and go to 'show-math-as' and then 'TeX Commands' to figure out the desired answer!* $\boxed{4}$.

Obviously this isn't ideal, as it allows students to just find and copy the answer code into the answer box without doing the problem. There are reasons to actually want this to be the case,[1] but most instructors would prefer to disable this feature.

Unfortunately we can't *literally* disable it as a feature without killing all the accessibility features of the page - which is obviously not something we want to do. Fortunately however, there are ways to disable it in practice.

First Solution using just LaTeX

The key to understanding how to fix this, is to realize exactly what is being exposed to the student. Specifically, the *content of the answer command* is what gets displayed to the student. As a result, it is as simple as changing what

Learning outcomes:

[1] It gives a known trap to funnel cheaters into and detect them for example

is in the actual answer command. The easiest way to manage this, is to declare a new command that contains the answer and using that instead of the answer itself. For example:

Problem 2 *Right-click the following answer box to find out the answer!* 3!

Feedback(*`attempt`*): *Ha, got you! The answer is actually 6.*

The code for the above problem is:

```
\begin{problem}
    Right-click the following answer box to find out the answer! $\answer{\ansOne}$
    \begin{feedback}
        Ha, got you! The answer is actually 6.
    \end{feedback}
\end{problem}
```

However, the key that makes it work is the hidden previous line, which is `\newcommand{\ansOne}{3!}`. Notice that I could actually leave the contents unsimplified (indeed, it has 3! and Xronos still took 6 as the answer). This can be used to obfuscate even more if you feel so inclined, but there isn't really any reason to do so.

Also note that you can (and should) define the answer command in the *body* of the document (after the begin document command) since the preamble is largely destroyed during the compiling of the assignment.

Potential Pitfalls and Problems

It is worth a note that you should have unique answer commands/macros for each answer box to avoid unexpected behavior (technically it *should* expand answers as they occur, but in practice you can run into some interesting expansion scope and timing quirks of Xronos here, so it's best to avoid it).

Also, ideally you want to avoid defining commands that depend on other custom commands. For example you want to avoid something like: `\newcommand{\ansTwo}{3 + \ansOne}`, writing instead `\newcommand{\ansTwo}{3 + 3!}`. Technically one can use custom commands within an answer command, but it almost certainly needs to use an immediate expansion macro definition instead, which would require the use of something like `\edef` or `\let` commands. If you don't recognize the difference between something like `\edef` and `\def` for LaTeX it is *highly* recommended you just avoid using custom commands inside answer commands in general.

Second Solution using Sage

Another option, which arguably has a number of other benefits, is to use sage to generate the problem and the answer. For example, consider the following problem:

Problem 3 *Let* $f(x) = 2\,x + 4$. *Then* $f(3) =$ $\boxed{10}$

Feedback(`attempt`***):*** *Right click the answer box and see what it shows...*

The above code is generated by:

```
\begin{sagesilent}
    p1c1 = 3
    p1f1 = 2*x + 4

    p1ans = p1f1(x=p1c1)
\end{sagesilent}
\begin{problem}
    Let $f(x) = \sage{p1f1}$. Then $f(\sage{p1c1}) = \answer{\sage{p1ans}}$

    \begin{feedback}
        Right click the answer box and see what it shows...
    \end{feedback}
\end{problem}
```

The obvious upside here is that you can then use sage to generate randomized and dynamic content in a nice way. If you are interested, you can check the advanced features section for more information on using sage, and best-practices and associated potential pitfalls and problems with sage.