The hmk package*

rapidcow
<thegentlecow@gmail.com>

September 12, 2022

Abstract

This describes a dumb little package I wrote just to highlight the answers in my homework assignments. (And some more, perhaps.)

Contents

1 List of required packages

The following packages are required:

amsmath, enumitem, keyval, xcolor, xparse

2 The \highlight command

`\highlight` Let's start with the most important macro here. At first we only had the `\hl` macro, which has only one argument, the text, and highlights it with a yellow color. But in this package we have a more general macro, namely `\highlight`. The syntax for the macro is

*This document corresponds to hmk v0.3b, dated 2021/06/09.

`\highlight[`⟨*setup*⟩`]{`⟨*color*⟩`}{`⟨*text*⟩`}`

so for example, `\highlight{yellow}{Hello!}` produces Hello! Optionally, you can provide configurations for the macro. One thing you can do is to change the margin around the highlight by using the key `hlsurround`, so

```
\highlight[surround=0pt]{yellow}{Hello!}
```

would produce Hello! Which you can see has no margin. Notice that you have to drop the "hl" prefix and type `surround` instead of `hlsurround`. This is so that we can have a special namespace for `\hl` commands. See Section 5 for the available options in this package (which aren't many—really. I'm too lazy to code all of these.)

And of course it shall certainly work in math mode! Let's say you have this wonderful sequence of calculation, then up to the *final* step you write:

> And so, by the question and our theory of work and energy, the gravitational potential energy of the ball
>
> $$GPE = mgh = 4.00\,\mathrm{kg} \cdot (9.81\,\mathrm{m/s^2}) \cdot 10.0\,\mathrm{m} \approx \text{392 J}.$$

where the math part can be (I'm sure you all can type the text part... right?) produced by

```
\[
  \mathit{GPE} = mgh = 4.00\, \mathrm{kg}
    \cdot (9.81\, \mathrm{m/s^2})
    \cdot 10.0\, \mathrm{m}
  \approx \highlight{cyan!30}{392\, \mathrm{J}}.
\]
```

Although, more often than not I would use the siunitx package. So the code would actually look like

```
\sisetup{per-mode=symbol}% Make the unit m/s^2 instead of ms^{-2}
\[
  \mathit{GPE} = mgh = \SI{4.00}{\kilo\gram}
    \cdot (\SI{9.81}{\meter\per\second\squared})
    \cdot \SI{10.0}{\meter}
  \approx \highlight{cyan!30}{\SI{392}{\joule}}.
\]
```

Do note that this package does not load siunitx, so you will have to load it in your preamble for this to work. (And it's not because I don't love it, just that it's not a dependency for this package...)

`\hl` This produces highlighting with a similar vibe of the once familar `\hl` in my past physics homework!

But first, I'd be dishonest if I didn't tell you that the name of this macro is directly taken from the package soul. Nevertheless, I did try the package but since I realized it doesn't work in math mode, I had to write one on my own, which is why I said this is reminiscent of the original `\hl` I wrote.

But anyway, to use it just simply call it. Literally! Like for example

```
\hl{Hello!!}
```

gives Hello!! Optional arguments are also allowed, so you can write

```
\hl[surround=0pt]{Hello!!}
```

in a similar fashion to the example we saw for `\highlight` to get Hello!! With no margin, that is...

Also in general the syntax looks like

`\hl[`⟨*setup*⟩`]{`⟨*text*⟩`}`

(Also if you want to know where the *exquisite* color came from... it is internally defined as `hlGold`. So essentially, `\hl` is just a short way to type `\highlight{hlGold}`.)

2.1 A technical note

Internally the `\highlight` just uses `\colorbox`, which in turn uses `\mbox`, meaning that no line wrapping will happen if your solution spans multiple lines and TEX might issue overfull box warnings if it can't fit the box onto one line. I haven't implemented a solution that can deal with line breaks (like the underline macros in the ulem package).

2.2 Defining custom highlight commands

`\newhighlight`
`\renewhighlight`
`\providehighlight`
`\declarehighlight`

You can define your own highlight command with any one of the command listed to the left. The general syntax is

`\newhighlight{`⟨*command*⟩`}[`⟨*color*⟩`]{`⟨*setup*⟩`}`

(Does this remind you of the `\newcommand` syntax?)

When ⟨*color*⟩ is specified, the defined macro takes in one mandatory argument: text. When ⟨*color*⟩ is omitted, the defined macro takes in two mandatory arguments: color, text. In either way, an optional argument can be applied to the defined macro to override the configuration.

As an example, the macro `\hl` itself is defined with `\declarehighlight`! It is defined with

```
\declarehighlight{\hl}[hlGold]{}
```

so that every time you call `\hl`, it uses the color `hlGold` and no configuration (and thus using default settings).

As another example, we can define

```
\newhighlight{\hlthick}{surround=1.2em}
```

and a thicker version of the default command without altering the settings So if you call `\hl`, you get this, but if you call `\hlthick{cyan!30}`, you get this.

3 Aligned \highlight

\Ahighlight For example

$$\begin{aligned} \colorbox{yellow}{$f(x) = \int h(x)\,\mathrm{d}x$} \\ &= g(x). \end{aligned}$$

may be produced with

```
\begin{align*}
  \Ahighlight{yellow}{f(x) &= \int h(x)\, \mathrm{d}x} \\
                           &= g(x).
\end{align*}
```

\Ahl Another shortcut for our *exquisite* color (okay I should stop saying that)

3.1 Defining \Ahighlight commands

\newAhighlight \renewAhighlight \declareAhighlight \provideAhighlight This works exactly like \newhighlight and others, so I won't repeat. The syntax is

\newhighlight{⟨*command*⟩}[⟨*color*⟩]{⟨*setup*⟩}

4 Miscellaneous commands

\pheq **Deprecated.** A simple macro cannot solve problems like these. It is advised that you stop using \pheq and its related options!

Equivalent to &\hphantom{{}={}}, thus the macro name, "**ph**antom **eq**uals". Made this because I got tired of typing it in front of the split and align environments (and it also looks unpleasantly long in my code).

\recip \drecip \trecip \recip#1 typesets \frac{1}{#1} exactly. For example:

$$T = \frac{1}{2\pi}\sqrt{\frac{m}{k}}.$$

may be produced with

```
\[
  T = \recip{2\pi}\, \sqrt{\frac{m}{k}}.
\]
```

Similarly, \drecip and \trecip do the exact same thing as \recip, except they call \dfrac and \tfrac (which are provided by amsmath).

5 Configurations

\hmksetup The usage of this macro is \hmksetup[⟨*prefix*⟩]{⟨*config*⟩}, where in ⟨*config*⟩ you would provide comma-separated list of ⟨*key*⟩=⟨*value*⟩ pairs. So for example,

```
\hmksetup{hlsurruond=0pt,hlleftmargin=1ex}
```

would set the key `hlsurround` to `0pt` and set the key `hlleftmargin` to `1ex`.

The ⟨*prefix*⟩ prepends keys with a certain name, so the above code can actually be typed as

```
\hmksetup[hl]{surround=0pt,leftmargin=1ex}
```

I have no idea why I came up with this... probably to give a fake sense of namespace (but also to restrict the user from calling irrelavent settings like `pheqafter` when using `\hl`! And it also means we can just reserve one family `hmk` while having separate "namespaces" in it, so... that's that.)

`\hlsetup` Equivalent to `\hmksetup[hl]{#1}`; this takes in just one argument. One thing to note is that `\hlsetup` is used by `\highlight`—namely when you provide the optional ⟨*setup*⟩ argument (see Section 2), `\highlight` passes the argument to `\hlsetup`. So if you write

```
\hl[surround=0pt]{Hello!}
```

it would call

```
\hlsetup{surround=0pt}
```

and then typeset the highlight box. (Of course, in a local scope so that your original configuration won't be overridden.)

And here are the keys and their appropriate description (I'm too lazy to format this haha kill me)

"Default" refers to the inital setup while loading the package. On the other hand, "keyval default" is means the default argument when you specify nothing. Usually this is empty, so that only specifying the key without the value is an error. But when this is filled (which is very unlikely), you can specify *just* the key and then when you use one of the setup commands, you assign that default value to that key.

Key	Description	Default	keyval default
`hlsurround`	The margin around `\highlight`	`0.5ex`	
`hlleftmargin`	Space to the left of the highlight box.	`0pt`	
`hlrightmargin`	Space to the right of the box.	`0pt`	
`hlabovemargin`	Space above the box.	`0pt`	
`hlbelowmargin`	Space below the box.	`0pt`	
`pheqbefore`	Space inserted before the equal sign.	`\thickspace`	
`pheqafter`	Space inserted after the equal sign.	`\thickspace`	

6 Bugs

Somehow using `\colorbox`, the highlighted area becomes too small in some cases with `surround=0pt` (which means that `\fboxsep=0pt` while the typesetting takes place). For example,

$$\zeta(s) = \frac{1}{1 - \frac{1}{p^s}}.$$

produced with

```
\[
  \zeta(s) = \hl[surround=0pt]{\recip{1 - \recip{p^s}}}.
\]
```

The "1" in the numerator seems to be sticking out of the box a little. (Does it have to do with `\fboxrule`?)

7 Implementation

Much of the code doesn't need comments, but for learning purposes I will be excessively verbose just so I, and potentially other people, can take note of some of the tricky parts of this code.

```
⟨*package⟩
```

Several packages we require for this package.

```
\RequirePackage{amsmath,enumitem,keyval,xcolor,xparse}
```

`\hl@surround` `\hl@abovemargin` `\hl@belowmargin` `\hl@leftmargin` `\hl@rightmargin`

Define a bunch of lengths.

```
\newlength{\hl@surround}
\newlength{\hl@abovemargin}
\newlength{\hl@belowmargin}
\newlength{\hl@leftmargin}
\newlength{\hl@rightmargin}
```

Set the default value—which is, only one of them. The rest are all just going to be `0pt` (as if they don't exist, ha!)

```
\setlength{\hl@surround}{0.5ex}
```

Now we also need a temporary box and length for `\highlight`. A temporary length is needed instead of `\@tempdima` and such because `\rule` internally uses them. Maybe one day we can use the TEX primitive `\vrule` instead, but... that's a job for another day.

```
\newsavebox\hmk@tempboxa
\newlength{\hmk@tempdima}
```

Define keys under families `hl` and `hmk` for the lengths.

```
\def\hmk@define@length#1{
  \define@key{hmk}{hl#1}{\setlength{\@nameuse{hl@#1}}{##1}}
}
```

Now set up the keys.

```
\hmk@define@length{surround}
\hmk@define@length{abovemargin}
\hmk@define@length{belowmargin}
\hmk@define@length{leftmargin}
\hmk@define@length{rightmargin}
\let\hmk@define@length\relax
```

`\hmksetup` User interface for setting keys under families `hl` and `hmk`. In reality, we only require one namespace and one command `\hmksetup` that has access to it all.

That being said, we make `\hmksetup` have two functions. One is to simply to be used as an interface to call `\setkeys`, and the other is to call `\setkeys`, but with every key prefixed by a specified name.

```
\long\def\hmksetup{%
  \@ifnextchar[%
    {\hmk@setkeys}{\setkeys{hmk}}%
}
```

`\hmk@setkeys` Here everything will be done in a similar fashion of `\setkeys`. One thing to note is that `\hmk@prefix` refers to the string appended to the keys, and not the family (first argument of `\setkeys`); the family is defined as `\hmk@family`.

```
%% Code borrowed from 'keyval.sty'.
\long\def\hmk@setkeys[#1]#2{%
  \def\hmk@family{hmk}%
  \def\hmk@prefix{#1}%
```

Start collecting the arguments. A comma is placed after `#2` to separate from the list provided by the user. `\relax` serves as a sentinel, which we will use to tell the end of iteration.

```
  \hmk@do#2,\relax,}
```

`\hmk@do` Now we will execute the iteratoin.

```
\long\def\hmk@do#1,{%
```

You might say that `\@empty` was redundant. Well I could've said so too! But just picture the moment when `#1` is empty. `\ifx\relax\else` is sure gonna evaluate to false and thus stopping the recursion. Now with `\@empty`, this if-statement is guarenteed to be false even when `#1` is empty, since `\relax` and `\@empty` are not equivalent (for reasons I can't quite explain).

```
  \ifx\relax#1\@empty \else
```

Now we have to split this arguments into parts. This is a bit trickier to explain so I won't expand on that now. One thing to note is that `\relax` is used as a delimiter for the end of argument of `\hmk@split`.

```
    \hmk@split#1==\relax
```

Honestly I still don't know why `\expandafter` is necessary here. A similar trick is used for macros like `\@ifnextchar` and `\@for` I believe, but I don't get it.

```
    \expandafter\hmk@do
  \fi
}
```

`\hmk@split` Now we split into parts. Similar to how `\Ahighlight` works, this is split into three parts, `#1`, `#2`, `#3`. Only the `#1` and `#2` are what we're going to use as the key-value pair, actually. `#3` is all the tokens that's left after two `=` have been scanned and right before a `\relax` token (which is placed as a sentinel in `\hmk@do`). We will discuss what each part contains as we progress.

```
\long\def\hmk@split#1=#2=#3\relax{%
```

Now we will examine `#1`. Of course, if `#1` is empty, then it means that the entire argument doesn't have any non-space token. In this case we simply do nothing. (`\hmk@@sp@def` strips away any leading or trailing space, I guess—that part of code is what I don't understand)

```
  \hmk@@sp@def\hmk@tempa{#1}%
  \ifx\hmk@tempa\@empty \else
```

Now we want to know if zero or at least one `=` is provided, the former implying us to pass only the key, and the latter implying we need to pass key-value altogether.

This can be determined by `#3`, which is only empty when no `=` is provided. This is because when one `=` is provided, `\hmk@split` will gather that `=` and the first `=` in `\hmk@do`, but *not* the second `=`. This leaves `#3` to collect that `=` and so `#3` is not empty anymore.

keyval uses the following construct to check the emptiness of `#3`, which... I won't modify.

```
  \ifx\@empty#3\@empty
    \setkeys{\hmk@family}{\hmk@prefix\hmk@tempa}%
```

If #3 is not empty, we can just go straight ahead and take #1 and #2, which are definitely filled with the key and value—in fact this is what keyval does, and you can try this with something like \setkeys{⟨*family*⟩}{a=1=2} and notice that =2 is ignored.

However here we will explicitly check for the case where the user has provided more than one =, in which case the second = the user provided will cause #3 to end up containing at least two =: the two = from \hmk@do. So we check whether #3 contains exactly one = and issue a warning if it is not.

```
  \else \def\@tempa{#3}\def\@tempb{=}%
    \ifx\@tempa\@tempb \else
      \PackageWarning{hmk}{Extra `=' is ignored}%
    \fi
    \setkeys{\hmk@family}{\hmk@prefix\hmk@tempa=#2}%
  \fi \fi
}
```

Now \@tempa would be a temporary macro that stores the definition. \@tempa will be called later to replace every #1 with a space token.

```
\def\@tempa#1{%
```

\hmk@@sp@def This macro just removes space in the fashion of keyval. In this case we only need to remove the space before \hmk@prefix, but I'm a bit too lazy to code that (and it's somewhat hard to for my current knowledge of TEX). Instead here is the source code directly copy-and-pasted.

```
\long\def\hmk@@sp@def##1##2{%
  \futurelet\hmk@tempa\hmk@@sp@d##2\@nil\@nil#1\@nil\relax##1}%
```

Refer to the documentation of keyval for the rest.

```
\def\hmk@@sp@d{%
  \ifx\hmk@tempa\@sptoken
    \expandafter\hmk@@sp@b
  \else
    \expandafter\hmk@@sp@b\expandafter#1%
  \fi}%
\long\def\hmk@@sp@b#1##1 \@nil{\hmk@@sp@c##1}%
  }
\@tempa{ }
\long\def\hmk@@sp@c#1\@nil#2\relax#3{\toks\z@{#1}\edef#3{\the\toks\z@}}
%% End of code borrowed from `keyval.sty'.
```

\hlsetup \hlsetup is just basically calling \hmksetup but with "hl" as a prefix of every key.

```
\long\def\hlsetup#1{\hmksetup[hl]{#1}}
```

Because we're about to process the options, we need to define keys for pheq in advance.

```
\define@key{hmk}{pheqbefore}{\def\pheq@before{#1}}
\define@key{hmk}{pheqafter}{\def\pheq@after{#1}}
\hmksetup[pheq]{before=\thickspace,after=\thickspace}
```

Now there's a weird thing that I want to do here: passing the key-value package options to \hmksetup. The following code is due to a post by egreg on TEX Stack Exchange. (Maybe it'd be better to use kvoptions, but I really don't want to

separate the interface in package option and `\hmksetup`, so for now… this is the clumsy code we're working with.)

```
% \newif\ifhmk@use@enit
% \DeclareOption{noenit}{\hmk@use@enitfalse}
% \DeclareOption{enit}{\hmk@use@enit@true}

\DeclareOption*{%
  \begingroup\edef\x{\endgroup\noexpand
    \hmksetup{\expandafter\noexpand\CurrentOption}}\x}
% \ExecuteOptions{enit}
\ProcessOptions\relax
```

`\highlight` The optional argument `#1` is the parameters passed to `\hlsetup`, `#2` is the color the highlight box is in, and `#3` is the text to be highlighted.

```
\DeclareDocumentCommand{\highlight}{ O{} m m }{%
```

As discussed in this answer, a `\relax` token is needed here to correctly identify whether we are in math mode if this command is placed right after `&` in alignment. In addition, we will start a group so that changes are local.

```
  \relax \begingroup
```

Setup. We will just let emtpy argument be, since `\hlsetup{}` does nothing.

```
  \hlsetup{#1}%
  \ifmmode
```

`\mathpalette` accepts two arguments and ultimately expands to `#1`⟨*math style*⟩`#2` (here `#1` and `#2` are the arguments of `\mathpalette`). Here the resulting call is `\hl@do{#2}`⟨*math style*⟩`#3`.

```
    \mathpalette{\hl@do{#2}}{#3}%
  \else
```

Similar to the code above for math mode, we pass the same arguments, but instead us a `\relax` token to distinguish text mode.

```
    \hl@do{#2}\relax{#3}%
  \fi \endgroup
}
```

`\hl@do` Argument `#1` is the color of the highlight box, `#2` is either a math style if we're in math mode (one of `\displaystyle`, `\textstyle`, etc.) or `\relax` if we're in text mode. `#3` is the text.

```
\def\hl@do#1#2#3{%
```

First, depending on whether we're in math mode or text mode, we will set the box differently. This is distinguished by checking `#2` is `\relax`. (**Question**: Why does `\ifx` work, but not `\if`??)

```
  \ifx#2\relax
    \setbox\hmk@tempboxa=\hbox{#3}%
  \else
```

Note: `\m@th` is equivalent to `\mathsurround=0pt`. egreg has a nice explanation of this.

`\m@th` seems to work in either way! (`$#2#3\m@th$` and `$\m@th#2#3$` both seem to work)

```
    \setbox\hmk@tempboxa=\hbox{$#2#3\m@th$}%
  \fi
```

Now we need to change `\fboxsep`, which is a dimen that is sort of like the margin of a colorbox.

```
  \setlength{\fboxsep}{\hl@surround}%
```

Now we will define a temporary macro that creates margin around the colorbox. `##1` is the sideways margin, `##2` is the margin above, and `##3` is the margin below.

```
  \def\hl@strut##1##2##3{%
```

Set `\hmk@tempdima` to the amount to dip below (the margin below), taking account of the `\fboxsep`, the depth of the box itself, and the margin below `##3`.

```
    \setlength{\hmk@tempdima}
      {\dimexpr\fboxsep + \dp\hmk@tempboxa + ##3\relax}%
```

Now for a very convenient method to create margin (recall that the call signature is `\rule[`⟨*raise height*⟩`]{`⟨*width*⟩`}{`⟨*thickness*⟩`}`).

```
    \phantom{\rule[-\hmk@tempdima]%
      {##1}{\dimexpr\hmk@tempdima + \ht\hmk@tempboxa + \fboxsep
        + ##2\relax}}%
  }%
```

Now define struts on the left and right.

```
  \def\hl@lstrut{%
    \hl@strut\hl@leftmargin\hl@abovemargin\hl@belowmargin}%
  \def\hl@rstrut{%
    \hl@strut\hl@rightmargin\hl@abovemargin\hl@belowmargin}%
```

Typeset the box. (Finally!) `\mbox` is used here to prevent the margin struts accidentally being displaced on a different paragraph.

```
  \mbox{\hl@lstrut\colorbox{#1}{\usebox\hmk@tempboxa}\hl@rstrut}%
}
```

`\Ahighlight` This macro takes in 3 arguments: `#1` is optional configuration passed to `\hlsetup`, `#2` is the color, and `#3` is the text.

This code is copied from `\Aboxed` in mathtools with slight alterations to adapt to our scenario. mathtools doesn't document this but I find the email archive it links to very helpful (specifically this).

It is worth noting that this macro assumes math mode with display style.

```
%% This macro is inspired by \Aboxed from 'mathtools.sty'.
\DeclareDocumentCommand{\Ahighlight}{ O{} m m }{%
```

Increase the master counter. The smart use of `\let` here prevents TEX from expanding it (in case the macro is placed right after `&`).

```
  \let\bgroup{\romannumeral-'}%
  \hmk@Ahl{#1}{#2}#3&&\@nil
}
```

`\hmk@Ahl` This is where we split up the arguments using `&`. The arguments `#1`, `#2` are just accepted as is, but starting from `#3` is where it gets tricky. The macro code demands `&` tokens, which are all provided by the call above in case the text doesn't contain any `&` token. However, when there is one `&` token, then the part before `&` is assigned to `#4` and the part after is assigned `#4`. Despite the two `&` remaining, the token `\@nil` makes sure that `#5` consumes anything that's left.

That being said, it *is* currently impossible for `\Ahighlight` to contain more than one `&`, since any more `&` would just end up in `#5` and ignored.

```
\def\hmk@Ahl#1#2#3&#4&#5\@nil{%
```

Decrease the master counter.

```
\ifnum0=`{}\fi
```

Load the configuration so we get the right width for `\hl@surround`.

```
\hlsetup{#1}%
```

Create a box with that will help us find the width of the box before the point of alignment. An empty math atom `{}` acts like an ordinary symbol, and is placed after `#3` to get proper spacing.

```
\setbox\z@=\hbox{$\displaystyle#3{}\m@th$\kern\hl@surround}%
```

Now take the aforementioned box and define a temporary macro that shifts things around. Note that since changes here are local and will be reset right after TEX reads the `&` token, we need to use `\the` to expand the dimen to pass on the width to the next column.

```
\edef\@tempa{\kern \wd\z@ &\kern -\the\wd\z@}%
```

Apply the shifting and typeset the box. Options are passed to `\highlight` fully.

```
\@tempa\highlight[#1]{#2}{#3#4}%
}
```

`\hmk@newhighlight`

`#1`: (internal use) xparse new command macro
`#2`: name of the new highlight macro
`#3`: color name
`#4`: highlight configurations for `\hlsetup`

```
\DeclareDocumentCommand{\hmk@newhighlight}{ m m o m }{%
```

First we check if the user is attempting to redefine `\highlight` (which is very dangerous!) There are some macros that we can safely dismiss that won't do any harm: `\NewDocumentCommand` will issue error for any macro already defined, and `\ProvideDocumentCommand` won't override any predefined macros either.

```
  \ifx#1\NewDocumentCommand
  \else \ifx#1\ProvideDocumentCommand
```

However, if the new-command macro is anything else (which has the danger of overriding the definition of `\highlight`), we have to make sure that the command to be defined is not `\highlight`.

```
  \else \ifx#2\highlight
    \PackageError{hmk}{Cannot redefine \protect\highlight}
     {Commands like \protect\newhighlight\space internally use
      \protect\highlight\space to define the new macro, so if you define
      \protect\highlight, bad stuff will happen... (infinite recursion)%
      \MessageBreak}
  \fi \fi \fi
```

Now we need to branch off accordingly to whether the color parameter is provided. If it is provided, then define a macro with one mandatory argument: text. Otherwise, define a macro with two mandatory arguments: color and text. In either case, an optional argument can be provided to override the setup defined in `#4`.

```
  \IfNoValueTF{#3}
    {#1{#2}{ O{} m m }{\highlight[#4,##1]{##2}{##3}}}%
    {#1{#2}{ O{} m }{\highlight[#4,##1]{#3}{##2}}}%
}
```

\newhighlight Provide public versions of \@newhighlight, each corresponding to an xparse macro.

```
\def\newhighlight{\hmk@newhighlight\NewDocumentCommand}
\def\renewhighlight{\hmk@newhighlight\RenewDocumentCommand}
\def\providehighlight{\hmk@newhighlight\ProvideDocumentCommand}
\def\declarehighlight{\hmk@newhighlight\DeclareDocumentCommand}
```

\hmk@newAhighlight Now for a similar implementation for \Ahighlight.

```
\DeclareDocumentCommand{\hmk@newAhighlight}{ m m o m }{%
  \ifx#1\NewDocumentCommand
  \else \ifx#1\ProvideDocumentCommand
  \else \ifx#2\Ahighlight
    \PackageError{hmk}{Cannot redefine \protect\Ahighlight}
     {Commands like \protect\newAhighlight\space internally use
      \protect\Ahighlight\space to define the new macro, so if you define
      \protect\Ahighlight, bad stuff will happen... (infinite recursion)%
      \MessageBreak}
```

Why on Earth would *anyone* try to redefine \highlight *here*??? I'm tired of writing an error message here. Screw them.

```
  \fi \ifx#2\highlight
    \PackageError{hmk}{Cannot redefine \protect\highlight}
      {Nope.  You just can't.\MessageBreak}
  \fi \fi \fi
  \IfNoValueTF{#3}
    {#1{#2}{ O{} m m }{\Ahighlight[#4,##1]{##2}{##3}}}%
    {#1{#2}{ O{} m }{\Ahighlight[#4,##1]{#3}{##2}}}%
}
```

\newAhighlight Provide public interface.

```
\def\newAhighlight{\hmk@newAhighlight\NewDocumentCommand}
\def\renewAhighlight{\hmk@newAhighlight\RenewDocumentCommand}
\def\provideAhighlight{\hmk@newAhighlight\ProvideDocumentCommand}
\def\declareAhighlight{\hmk@newAhighlight\DeclareDocumentCommand}
```

We are fully irresponsible for the notorious definition of \@ifundefinedcolor; please blame xcolor.

That being said, this does define the *classic* yellow color that we all love. (This is essentially equivalent to "Gold!50" if you use the svgnames option of xcolor...)

(Actually now come to think of it... I still think it is necessary that we override the definition of hlGold, in case it was defined. Otherwise we're just gonna get a quirky color and users are not gonna like it.)

```
% \@ifundefinedcolor{hlGold}{
\definecolor{hlGold}{rgb}{1,0.844,0}
\colorlet{hlGold}{hlGold!50}
% }{}
```

\hl This is a shortcut for our lovely *lovely* color.

```
% \DeclareDocumentCommand{\hl}{ o m }{\highlight[#1]{hlGold}{#2}}
\declarehighlight{\hl}[hlGold]{}
```

\Ahl Just yet another shortcut for our *beautiful exquisite* color. We haven't got any \newAhighlight, so this should do the job.

```
% \DeclareDocumentCommand{\Ahl}{ o m }{\Ahighlight[#1]{hlGold}{#2}}
\declareAhighlight{\Ahl}[hlGold]{}
```

\pheq "Always helpful," as one wise man once said... who could it have been?

```
\def\pheq{&\pheq@before\hphantom{=}\pheq@after}
```

\recip Reciprocal. The latter two \drecip and \trecip require amsmath loaded.
\drecip
\trecip

```
\def\recip#1{\frac{1}{#1}}
\def\drecip#1{\dfrac{1}{#1}}
\def\trecip#1{\tfrac{1}{#1}}
```

Next we check whether the exam class is loaded.

```
\@ifclassloaded{exam}
  {
```

\setquestion For such \setquestion macros, we can expect invalid input to be complained by \numexpr... *except* when the input is empty. That's when we need to issue a warning ourselves.

Optionally we can check for the right level with defined by the class exam...

```
    \providecommand{\setquestion}[1]{%
      \@ifundefined{@checkqueslevel}{}{\@checkqueslevel{question}}%
      \def\@tempa{#1}%
      \ifx\@tempa\empty
        \PackageWarning{hmk}{%
          \protect\setquestion\space received empty argument;
          defaulting to 0.}
      \fi
      \setcounter{question}{\numexpr#1-1\relax}}
```

Ahh... i was wondering where the part counter could possibly be defined! It turns out that it was defined by article.cls! (The top level of headers above section) And so that's why exam.cls has to give this part counter a special name partno...

```
    \providecommand{\setpart}[1]{%
      \@ifundefined{@checkqueslevel}{}{\@checkqueslevel{part}}%
      \def\@tempa{#1}%
      \ifx\@tempa\empty
        \PackageWarning{hmk}{%
          \protect\setpart\space received empty argument;
          defaulting to 0.}%
      \fi
      \setcounter{partno}{\numexpr#1-1\relax}}
    \providecommand{\setsubpart}[1]{%
      \@ifundefined{@checkqueslevel}{}{\@checkqueslevel{subpart}}%
      \def\@tempa{#1}%
      \ifx\@tempa\empty
        \PackageWarning{hmk}{%
          \protect\setsubpart\space received empty argument;
          defaulting to 0.}%
      \fi
      \setcounter{subpart}{\numexpr#1-1\relax}}
    \providecommand{\setsubsubpart}[1]{%
      \@ifundefined{@checkqueslevel}{}{\@checkqueslevel{subsubpart}}%
      \def\@tempa{#1}%
      \ifx\@tempa\empty
        \PackageWarning{hmk}{%
```

```
        \protect\setsubsubpart\space received empty argument;
        defaulting to 0.}%
    \fi
    \setcounter{subsubpart}{\numexpr#1-1\relax}}
}{
```

`question` Otherwise, we will be making up environments for exam!

```
  \newlist{question}{enumerate}{5}
  \setlist[question,1]{label=\arabic*.}
  \setlist[question,2]{label=(\alph*)}
  \setlist[question,3]{label=(\roman*)}
  \setlist[question,4]{label=(\Alph*)}
  \setlist[question,5]{label=(\Roman*)}
```

`\setquestion` Oh yeah, and `\@listdepth` is a counter allocated by LaTeX that keeps track of the nested level of the `enumerate` environment. It seems that enumitem also uses it when defining its custom lists, so I put this here.

`\romannumeral` is a weird TeX primitive... I said weird because it somehow accepts both integers and counters (which is why I didn't have to put something like "`\the\@listdepth`" in the optional argument).

```
  \DeclareDocumentCommand{\setquestion}{ O{\@listdepth} m }{%
```

Like the `\setquestion` before... issue the same warning.

```
    \def\@tempa{#1}%
    \ifx\@tempa\empty
      \PackageWarning{hmk}{%
        \protect\setquestion\space received empty argument;
        defaulting to 0.}%
    \fi
    \setcounter{question\romannumeral#1}{\numexpr#2-1\relax}%
  }
}
```

⟨/package⟩

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.