Related Links:

Question Renderer:

- GitHub - drdrew42/renderer: PG Renderer for WeBWorK problems [powered by Mojolicious]
- GitHub - lpc0220/standalone-question-renderer: This isolates the features of webwork which simply render a problem, independent of any connection with a database or with a webserver

*** Authers Page Wiki:**

Authors - WeBWorK_wiki

Category:Sample Problems - WeBWorK_wiki

* Basic Perl syntax - WeBWorK_wiki

Sample Problem 2:

This sample shows how to write **three commonly** used problem types: **formulas** making use of **more of the MathObjects Formula functionality**, **multiple choice**, and **string entry problems**.

PG codes	Explanation

`# DESCRIPTION`
`# A simple sample problem that illustrates`
`# three common problem types.`
`# WeBWorK problem written by Gavin LaRose,`
`# <glarose(at)umich(dot)edu>`
`# ENDDESCRIPTION`
 	
`## DBsubject('WeBWorK')`
`## DBchapter('Demos')`
`## DBsection('Problem')`
`## KEYWORDS('')`
`## TitleText1('')`
`## EditionText1('')`
`## AuthorText1('')`
`## Section1('')`
`## Problem1('')`
`## Author('Gavin LaRose')`
`## Institution('UMich')`
	Tagging and description section
	
	All of the tagging information exists to allow the problem to be easily indexed.
	
	There is an on-line list of current chapter and section names and a similar list of keywords, as well as a page of best practices for tagging problems.
	
	Similar as sample 1, there's just only the ***comment*** section.

`DOCUMENT();`
`loadMacros(`
`"PGstandard.pl",`
`"MathObjects.pl",`
`"PGchoicemacros.pl",`
`);`
	Initialization section
	
	To use the **multiple choice object** we use in this problem, we have to add **PGchoicemacros.pl** to the list of macros files that we load.

`# make sure we're in the context we want`
`Context("Numeric");`
 	
`# INITIALIZATION FOR PART 1`
`# set up a function for our formula problem.`
`Context()->variables->are(t=>'Real');`
`$a = random(2,9,1);`
`$func = Formula("cos($a t)");`
`$funcDeriv = $func->D('t');`
`$m = $funcDeriv->eval(t=>2);`
`$y0 = $func->eval(t=>2);`
`$line = Formula("$m (t - 2) + $y0");`
	Problem set-up section
	
	For **part 1**, we define a function of the variable `t`. By default, the only defined variable is x, so we first add the real variable `t` to the Context. A list of commonly used Context modifications is available. Then we define the function, find its derivative, and work out the equation of the tangent line to the function.
	
	Note: we're using a number of characteristics of Formulas here. `$f->D('t')` finds the derivative of `$f` with respect to `t`.
	Then, `$f->eval(t=>2)` evaluates `$f` at the point `t=2`.
	Note that values must be specified for all variables when calling eval.
	
	If $f'(a)$ exists, then $\lim_{x \to a} f(x)$
	A. must exist, but there is not enough information to determine its value.
	B. is equal to $f(a)$.
	C. is equal to $f'(a)$.
	D. might not exist.
	E. does not exist.

```
# INITIALIZATION FOR PART 2
# set up for a multiple choice problem.
$radio = new_multiple_choice();
$radio->qa("This problem is", "easy");
$radio->extra("very easy", "hard");
$radio->makeLast("impossible");
```

For **part 2**, we define a **radio object** to include a multiple choice problem.

The `$radio->qa("question","correctAnswer")` line gives the **question** and **answer** to the question.
Then `$radio->extra("answer","answer",...)` defines **extra (incorrect) answers** for the problem.
By using `$radio->makeLast("lastAnswer")` we can **specify which answer is displayed last** (all others will be presented in a **random order**).
To make the correct answer be the last one, just call `$radio->makeLast("correctAnswer")` anytime after the initial `$radio->qa("question","correctAnswer")` call.

```
# INITIALIZATION FOR PART 3
Context()->strings->add(True=>{},False=>{});
$strAns = String('True');
```

For **part 3,** we are **asking the student to enter a String answer**. To avoid error messages we first add to the Context all valid strings (in this case, the strings "True" and "False"). Then we define a String object which gives the correct answer.

```
TEXT(beginproblem());
Context()->texStrings;

BEGIN_TEXT
${BBOLD}Part 1$EBOLD
$BR
Find the equation of the line tangent to
\(f(t) = $func\) at \(t=2\).
$BR
\(y = \) \{ ans_rule(35) \} (note that
this must be a function of \(t\).)

$PAR
${BBOLD}Part 2$EBOLD
$BR
\{ $radio->print_q() \}
\{ $radio->print_a() \}

$PAR
${BBOLD}Part 3$EBOLD
$BR
(Enter True or False: ) All instructors
love WeBWorK.  \{ ans_rule(6) \}

END_TEXT

Context()->normalStrings;
```

Text section

Note that we use a couple of **formatting variables** here: `${BBOLD}` begins a **bold section of text**, and `$EBOLD` ends it. `$BR` inserts a line break, and `$PAR` inserts a paragraph break.

For **Part 1**, we display the equation in-line with `\( f(t) = $func\ )` and then ask for the answer with an answer rule.

For **Part 2**, we can just use the **radio button object** to print the question and options. Recall that `\{ \}` executes code in the text section; `$radio->print_q()` (and `...print_a()`) are methods of the radio object that print the question and answers.

For **Part 3**, we just ask a question and insert an answer rule.

```
ANS( $line->cmp() );
ANS( radio_cmp( $radio->correct_ans() ) );
ANS( $strAns->cmp() );

Context()->texStrings;
SOLUTION(EV3(<<'END_SOLUTION'));
$PAR SOLUTION $PAR
${BBOLD}Part 1$EBOLD
$BR
The derivative of \(f(t) = $func\) is
\(f'(t) = $funcDeriv\), so the slope of
the line is \(f'(2) = $m\).  Then when
\(t = 2\) we have \(f(2) = $y0\), so the
equation of the line is \(y = $line\).

$PAR
${BBOLD}Part 2$EBOLD
$BR
This problem is clearly easy.

$PAR
${BBOLD}Part 3$EBOLD
$BR
A careful study of one WeBWorK
instructor showed that 100% of all
respondents thought this statement
was true.
END_SOLUTION
Context()->normalStrings;

ENDDOCUMENT();
```

Answer and solution section

Because there are **three parts** to the question, we have three `ANS()` calls. These evaluate the answer blanks in the order that they appeared in the problem.

For the **first part** of the problem, we want to check that the student entered the right equation for the line. We therefore use the comparison method of the `$line` Formula we defined.

For the **second part** of the problem we have to check that the student entered the correct (multiple-choice) answer. At the moment this requires that we use an *old-style answer checker*, radio_cmp. This just verifies that the answer that the student submitted is the same as the correct answer to the problem.

For **part 3**, we just check that the answer the student submitted compares with the expected String.

Finally, note that we can use any formatting that we use in the text section of the problem in the solution.