Related Links:

* **Authers Page Wiki:**

Authors - WeBWorK_wiki
Category:Sample Problems - WeBWorK_wiki

* SampleProblem4 - WeBWorK_wiki

Sample Problem 3: A PGML WeBWorK Sample Problem

This sample problem illustrates the **basics** of how to use **PGML commands** to layout a question.

As usual a **standard WeBWorK PG file** has **five sections**:

1. A **tagging and description section**, that describes the problem for future users and authors,
2. An **initialization section**, that loads required macros for the problem,
3. A **problem set-up section** that sets variables specific to the problem,
4. A **text section**, that gives the text that is shown to the student, and
5. OPTIONAL --An **answer section**, that specifies how the answer(s) to the problem is(are) marked for correctness, and gives a solution that may be shown to the student after the problem set is complete. **As you will see this section can be used but are not necessary when using PGML commands.**
6. A **solution section**

What is PGML?

PGML **(Problem Generation Markup Language)** is a **simplified markup syntax** for writing WeBWorK problems. It is **built on top of the traditional PG language** but provides a **cleaner, more intuitive way** to format problem text, define answer blanks, and embed dynamic content. Think of PGML as a **"user-friendly wrapper"** for PG, designed to **reduce boilerplate code** and **improve readability**.

PG codes	Explanation
# DESCRIPTION # A simple sample problem that asks students to # enter a bunch of different types of answers # WeBWorK problem written by Gavin LaRose # <glarose(at)umich(dot)edu> # and modified by Mike Gage gage(at)math(dot)rochester(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", **"PGML.pl",** "PGcourse.pl",);	**Initialization section** The first executed line of the problem **must** be the DOCUMENT(); command. Note that every command must end with a semicolon. We load the **PGML.pl** file to load the **PGML formatting commands** (similar to **markDown**).

# make sure we're in the context we want 	
Context("Numeric");
 	
$showPartialCorrectAnswers = 1;
$f = Formula("cos^2(x)+sin^2(x)");
	Problem set-up section
	
	Context("Numeric"); sets the "**context**", which determines **how variables are interpreted**. Contexts and context explanations are given on this help page. **(No points, vectors, matrices, complex numbers, or intervals are allowed.)**
	
	-- 	
	$showPartialCorrectAnswers controls whether students see feedback for **individually correct answers** in a problem **before they fully solve it**. When enabled (**= 1**), students see which answers are correct and which are incorrect. When disabled (**= 0**), **no per-**

answer feedback is given until the **entire problem is correct**.

For example:
Suppose the student enters:

- First blank: **2** (correct)
- Second blank: **4** (incorrect)

When $showPartialCorrectAnswers = 1:
Feedback:

✧ The first answer (**2**) is marked **correct** (e.g., green checkmark).

✧ The second answer (**4**) is marked **incorrect** (e.g., red X).

When $showPartialCorrectAnswers = 0:
Feedback:

✧ Both answers are marked **incorrect** (even though the first is correct).

All **scalar variables** are prefaced with a **dollar sign**: thus `$a` is a **variable** that has a **(non-array, non-associative-array)** value.

```
TEXT(beginproblem());

BEGIN_PGML
The number twelve is [_______]{12}
Type the  formula [`1+\frac{x}{2}`]
[__________]{"1+x/2"}

Twelve is [______]{Real(12)}
2 mod 10 is
[______]{Real(2)->with(period=>10)}
[`[$f]`] is equal to [_______]{Real(1)}
Twelve is [______]{num_cmp(12)}

The number 12 is
[____]{answer=>12,width=>10}
END_PGML
```

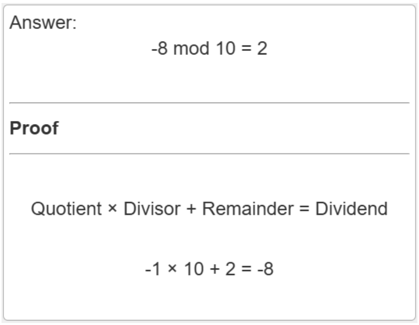
Text section

The `TEXT(beginproblem());` line displays a **header** for the problem.

Everything between `BEGIN_PGML` and `END_PGML` (each of which must appear alone on a line) is shown to the student.
`BEGIN_PGML` and `END_PGML` replace
the `BEGIN_TEXT`/`END_TEXT` structure used in older-style template samples.
The `Context()->texStrings` seen in those samples line is **not needed when using PGML**.

Answer blanks are indicated by [_______] where **the number of blanks indicates the width of the answer blank**. The **correct answer** can be given in curly braces immediately afterward `{"1+x/2"}`.

TeX formulas within the text of the problem can be entered as ``[`1+\frac{x}{2}`]``. A **variable** substitution would be given as `[$a]`,
while ``[`[$f]`]`` typesets the formula for `$f` in inline math mode.

	1. {Real(2)->with(period=>10)} ✧ Real(2) creates a **MathObject** expecting a numerical answer of **2**. ✧ ->with(period=>10) specifies that the answer is **periodic with a period of 10**. This means **any number equivalent to 2 modulo 10 (e.g., 2 modulo 10, 12 modulo 10, -8 modulo 10, all results are same, which is 2) will be accepted**. This is used for answers like "**2 mod 10**."  Answer: -8 mod 10 = 2 **Proof** Quotient × Divisor + Remainder = Dividend -1 × 10 + 2 = -8 2. {num_cmp(12)} ✧ **num_cmp** is a legacy answer checker for numerical answers. It verifies if the student's input matches the expected value (**12** in this case) within a **default tolerance**. While still functional, **num_cmp** is **less flexible** than **MathObjects** (e.g., **Real(12)**), which are preferred for better error handling and customization. 3. {answer=>12,width=>10} ✧ answer=>12 sets the **correct answer** to **12**. ✧ width=>10 adjusts the **width of the input field** in the **HTML form to 10 characters**. This affects **display only**, not answer validation.
BEGIN_PGML_SOLUTION You can use PGML in your solution if you use the structure above. There is currently no short cut. END_PGML_SOLUTION ENDDOCUMENT();	~~Answer and~~ **Solution** section Since the **answers were given alongside the problems** when you use **PGML**, the **answer section is not needed, although it is allowed**. Then, we explain the **solution** to the student. You can use BEGIN_PGML_SOLUTION/END_PGML_SOLUTION just as you would BEGIN_SOLUTION/END_SOLUTION **if you were not using PGML**. There is also BEGIN_PGML_HINT/END_PGML_HINT for **providing a hint** to the student.

	The ENDDOCUMENT(); command is the last command in the file.

Key Differences Between PG and PGML

Feature	PG (Traditional)	PGML
Syntax	Uses **Perl-based syntax** with `BEGIN_TEXT`/`END_TEXT` blocks and **escaped variables (转译变量)**.	Uses **markdown-like syntax** with `[` and `]` for **variables/answer blanks**.
Answer Blanks	Requires `ans_rule(width)` or `ANS(...)` macros.	Uses `[_____]` for answer blanks (**automatically numbered and linked to answer checkers**).
Variable Insertion	Variables **must be escaped**: `\($var\)` or `\{$var\}`.	Variables **embedded directly**: `[$var]`.
Formatting	Requires **HTML-like** tags (e.g., **$BR**, **$BOLDtext$EBOLD**).	Supports **markdown-like** formatting (e.g., **bold**, *italic*, lists).
Code Readability	**Cluttered** with Perl code and escaped variables.	**Clean separation** of text and logic; closer to natural writing.
Answer Checkers	Defined separately using `ANS(...)` or `ANS($answer->cmp)`.	Inline answer checkers: `[@ $answer->cmp @]`.
Context Handling	Requires explicit `Context()` declarations.	Inherits the current context but allows local modifications.

Clear comparison:

PG (Traditional PG) Style	Same in PGML Style

DOCUMENT();
loadMacros("PGstandard.pl",
"MathObjects.pl");
 	
Context("Numeric");
$f = Formula("x^2");
$dfdx = $f->D('x');
 	
TEXT(beginproblem());
BEGIN_TEXT
Find the derivative of \(f(x) = $f \).
$BR
\(f'(x) = \) \{ ans_rule(20) \}
END_TEXT
 	
ANS($dfdx->cmp);
ENDDOCUMENT();
	DOCUMENT();
	loadMacros("PGstandard.pl",
	"MathObjects.pl", "PGML.pl");
	
	Context("Numeric");
	$f = Formula("x^2");
	$dfdx = $f->D('x');
	
	BEGIN_PGML
	Find the derivative of [`f(x) = [: $f :]`].
	
	[`f'(x) =`] [_____]{$dfdx}
	END_PGML
	
	ENDDOCUMENT();

Markdown-Like Formatting:

```
BEGIN_PGML
**Bold Text**
*Italic Text*
- List item 1
- List item 2
END_PGML
```

Conditional Text:

```
BEGIN_PGML
[% if $is_correct %]Correct![% else %]Try again.[% END %]
END_PGML
```

Multi-Line Math:

```
BEGIN_PGML
[`\begin{align*}
  f(x) &= x^2 \\
  f'(x) &= 2x
\end{align*}`]
END_PGML
```