Intro to WeBWorK
Problem Authoring

Andrew Parker
NYC College of Technology

0.

Fundamentals of Perl

Syntax and structure

“ A novice had a problem and could not find a solution. “I know,” said the novice, “I’ll just use Perl!” The novice now had two problems.

-Erik Naggum

Seriously, just the basics

Variables

Variables are named with a leading dollar-sign.

```
$tau = 6.28318530718;
$string = “Twice pi is $tau.”;
```

Lists

Define a list-variable using @ instead of $.

```
@myList = (1,2,3,4);
```

Indices start at zero.

`$myList[1]` would be 2

Lines

At the end of each expression that will be executed, you must use a semicolon;

Comments

Please leave notes about the code you write. Anything that comes after a # is just there for explanation.

Conditionals

“If” statements use a logical statement and are followed by a code block {}. Use “elsif” to follow up with another logical test and code block.

For the **bold**

Learn Perl in about 2 hours 30 minutes

https://qntm.org/perl_en

1.

Problem Structure in PG

Framework and overview

There has been ~25 years of evolution

- Many authors get their start by modifying OPL problems
- OPL problems are growing older
- Metadata and tagging are timeless

Just be aware that Library problems span several generations of problem authoring frameworks.

Two major leaps forward

MathObjects

“Objects” that have the ability to represent themselves in different formats, and know how they should be compared to other objects.

PGML

Adoption of markdown language as a framework for formatting problem content in a standardized and easy-to-use way.

A problem structure overview

Metadata

Make sure your problem is properly tagged and classified so that it can be found by other users when shared publicly.

Setup

Build up the parameters that will eventually provide the content for your problem text, hints, and solutions.

Display

Write out the question, ask for student responses. Provide for optional hints and solutions.

2.

Building the framework

Working with Context and MathObjects

Start with the tags

- DBsubject, DBchapter, DBsection
- Find some relevant keywords
- Don’t forget to include yourself as author!

Tagging problems is the biggest pain -- especially if you’re writing a significant quantity of problems. Do this first!

Choosing the right Context()

Numeric

Commonly used for problems with answers that are Real numbers or Formulas. You can set which variables your formulas will use (as well as their domains).

Point

This context is used for problems with answers that are Points in R^n. You can set the dimension by adding more variables to the Context.

Complex

This context is for fans of algebraically closed fields - and characteristic zero, of course.

Vector

Similar to points, the vector context supports i, j, k notation, angle braces, and column-vector notation.

Matrix

The matrix context inherently supports determinants, transposes, inverses, and trace

Interval

This context supports unions of intervals, intersection, subsets, set-minus, reduces overlapping intervals, and it also supports finite sets of real numbers.

So you want to get fancy?

Non-standard Contexts

There are lots of additional contexts to choose from. These extra contexts must be imported in the loadMacros() call.

Configure

All contexts have configurable parameters that allow you to change the way your objects are represented or compared.

Customize

Write your own answer checker, or use other add-ons (like AnswerHints) to respond to common incorrect answers.

Let’s get random

- random(*min*, *max*, [*step*=1])
- list_random(1, 2, 3, 5, 8)
- Use do-while to enforce restrictions on randomization

Randomization can go off the rails very quickly. Controlling for this is much easier when you write with the solution already in mind.

Putting it all together

MathObjects

Combine your randomized parameters to make new MathObjects.

```
$f = Formula(“sqrt(x^2
- $c^2)”)->reduce;
```

Answer

Make sure you create an answer object as well!

```
$ans = Interval(“R -
(-$c, $c)”)->reduce;
```

Reduce

Make sure to use ->reduce on any formulas you plan on displaying in your problem. Answers too!

3.

Layout with PGML

What the problem actually *looks* like

PGML blocks don’t work as code

- Wrap your problem in BEGIN_PGML/END_PGML
- All contents of a PGML block are rendered as text
- BEGIN_PGML_HINT and BEGIN_PGML_SOLUTION too!

Do all of your randomization and MathObject construction outside of your PGML blocks.

Mark it down

Markdown

Markdown language is used to layout text on lots of websites. Follow the usual *bold*, _italic_, and

1. numbered lists

Use variables

Variables can be used in your markdown when you wrap them in square brackets.

```
[$variable]
```

Embed TeX

Math objects TeX themselves automagically.

~~~~
[`[$function]`]

[``[$function]``]

[```[$function]```]
~~~~

4.

Let’s write some code

Turn a static problem into a randomized one