Randomizing with Sage

Common Sage Functions and Details

Details when using Randomized Content with Sage

Despite having used sage in some of the previous pages, you might notice that none of it was actually randomized. Indeed, if you look at the top right corner of this page, you will see a green "Try Another" button, but in the previous pages, the button doesn't appear. This is because the "Try Another" button only appears if there is sage content that is being randomized.

How the 'Try Another' button works

When you use randomized values as part of the sage code, the green "Try Another" button appears. This button essentially re-rolls all the randomized values and repopulates the *entire* page. There is no way to only randomize part of the page's randomized code, it always rebuilds the entire page (there are weird pedagogical reasons to do things this way and this just happens to be how the Ximera team decided to go - but it does purposefully work this way).

If you click the "Try Another" button it may (it doesn't always) pop up a warning saying that all your work will be deleted and progress will be lost. It is noteworthy that, Canvas won't lower grades because the progress is reset on the Xronos page. However, in order to get full credit, the student needs to have everything completed *at the same time*. It is not sufficient to, say, do problems one through nine, then "Try Another" and then do problem 10 out of 10 to get full credit - they would have to redo all of one through nine as well to get 100% completion.

Also, as a deliberate pedagogical choice, all problems given to students use the same randomized seed - meaning all students get the *same* randomized problems in the same order. This is to allow students to work together on content, as well as for a bunch of technical reasons.

Useful Sage Reference Links

There are a couple useful links that have a bunch of built-in esoteric mathematical functions/ideas for sage. I figured I'd put them here, and add them as I

Learning outcomes:

find them.

- Sage Quickref on Linear Algebra
- Sage Quickref on Number Theory
- A comprehensive introductory course on Sage - course notes.

Common Functions to Use

Generally we include the following block at the top of the first sagesilent environment of a file that intends to use sage:

```
#####Define default Sage variables.
#Default function variables
var('x,y,z,X,Y,Z,r,R')
#Default function names
var('f,g,h,dx,dy,dz,dh,df')

def RandInt(a,b):
    """ Returns a random integer in [`a`,`b`]. Note that `a` and `b` should be integers
    """
    return QQ(randint(int(a),int(b)))
    # return choice(range(a,b+1))

def NonZeroInt(b,c, avoid = [0]):
    """ Returns a random integer in [`b`,`c`] which is not in `av`.
        If `av` is not specified, defaults to a non-zero integer.
    """
    while True:
        a = RandInt(b,c)
        if a not in avoid:
            return a
```

This defines the default (mathematical) variables for sage (more can be added if needed) and default (mathematical) function names, along with the RandInt and NonZeroInt functions.

RandInt

This generates a random integer between a and b (inclusive). So `p1c1 = RandInt(-5,5)` creates, and defines, the sage variables “p1c1” as a random integer between -5 and 5. This is as straight forward as it sounds.

NonZeroInt

NonZeroInt started as a version of RandInt to avoid zero, but even while writing the initial definition it was obvious that zero may not be the only number you may want to avoid - and indeed, you may want to avoid more than just one value. So NonZeroInt(-5,5) gives a non-zero integer between -5 and 5, but you can add a third (optional) argument which is a list of values to avoid. So `p1c2 = NonZeroInt(-5,5,[-p1c1,0,p1c1])` would create the sage variable "p1c2" and assign it a random integer between -5 and 5, but not whatever -p1c1 or p1c1 are, and not 0 as well. Note that `p1c3 = NonZeroInt(-5,5,[p1c1])` would create the sage variable p1c3, and assign it a random number between -5 and 5 - except for p1c1... and importantly, if p1c1 is not zero, then p1c3 *could* be assigned zero unless you include it in the list of things to avoid.

Example

Consider the following problem:

Exercise 1 *Is the following polynomial factorable?*

$$3\,x^2 + x - 4$$

Multiple Choice:

(a) *Yes* ✓

(b) *No*

The above problem is generated by the code:

```
\begin{sagesilent}
    #####Define default Sage variables.
    #Default function variables
    var('x,y,z,X,Y,Z,r,R')
    #Default function names
    var('f,g,h,dx,dy,dz,dh,df')

    def RandInt(a,b):
        """ Returns a random integer in [`a`,`b`]. Note that `a` and `b` should be i
        """
        return QQ(randint(int(a),int(b)))
        # return choice(range(a,b+1))
    def NonZeroInt(b,c, avoid = [0]):
        """ Returns a random integer in [‘b‘,‘c‘] which is not in ‘av‘.
            If ‘av‘ is not specified, defaults to a non-zero integer.
        """
        while True:
            a = RandInt(b,c)
            if a not in avoid:
                return a

    p1c1 = NonZeroInt(-10,10)
    p1c2 = RandInt(-5,5)
    p1c3 = RandInt(-5,5)
    p1f1 = expand( (p1c1*x-p1c2)*(x-p1c3) )
\end{sagesilent}
\begin{exercise}
    Is the following polynomial factorable?
    \[
        \sage{p1f1}
    \]
    \begin{multipleChoice}
        \choice[correct]{Yes}
        \choice{No}
    \end{multipleChoice}
\end{exercise}
```

You can hit the “Try Another” button above and see how the problem randomizes.