1. Log in to your Sage/CoCalc account.
   (a) Start the Chrome browser.
   (b) Go to \textcolor{blue}{http://cocalc.com} and sign in.
   (c) You should see an existing Project for our class. Click on that.
   (d) Click “New”, call it \textcolor{red}{c25}, then click “Sage Worksheet”.
   (e) For each problem number, label it in the Sage cell where the work is. So for Problem 1, the first line of the cell should be \textcolor{red}{#Problem 1}.
   (f) When you are finished with the worksheet, click ”make pdf”, email me the pdf (at \textcolor{blue}{clarson@vcu.edu}, with a header that says \textcolor{red}{Math 255 c25 worksheet attached}).

The Birthday Problem.

2. (Guess) \textbf{How many students do we need in a classroom so that there is a better than 50\% chance that at least one pair of them have the same birthday (Month & Day)?}

Here’s 3 basic and useful facts:

- **Multiplication Principle** (Burgers & Drinks). If there are \textcolor{red}{m} choices for one thing and \textcolor{blue}{n} choices of another thing, there are \textcolor{red}{m} \cdot \textcolor{blue}{n} ways to choose one from the first things and one from the second things.

- (Assuming all outcomes are equally likely) the probability of something happening is:
  $$\frac{\text{The number of good outcomes}}{\text{The total number of outcomes}}$$

- **Opposite Probabilities**. the probability of an event happening $= 1 -$ the probability it doesn’t happen.

3. Start by simplifying the problem. What is the probability that 2 students have the same birthday? This can be calculated directly: think of the students (or more clearly their birthdays) as \textcolor{red}{S1} and \textcolor{blue}{S2}. So the possible pairs of birthdays can be represented as (\textcolor{red}{S1}, \textcolor{blue}{S2}). If we assume a year always has 365 days, how many “good” pairs are there (pairs where the birthdays are the same). How many total pairs are there? What’s the probability?
4. Now what’s what is the probability that if there are 3 students at least 2 will have the same birthday? (This is harder and needs a new idea).

5. Now what’s what is the probability that if there are 4 students at least 2 will have the same birthday? If you get this you may see the pattern and write code to figure out an answer for any number \( n \) of students.

6. Then keep repeating. What’s the smallest \( n \) where the probability is finally .5 or 50%. (Write a while loop with an appropriate stopping condition or a for loop. We know that we will have guaranteed certainty when \( k = 365 \)).

7. How many students do we need to have a 90% chance that at least 2 students share the same birthday?

More Interacts!

There is a collection of examples of Sage INTERACTs at http://wiki.sagemath.org/interact/. Let’s look at a few of these examples to see the kinds of things you can do with Sage.

Posets

8. A poset is a very common mathematical object. They consist of a set together with a relation that is reflexive, transitive, and anti-symmetric. Any collection of lists or sets with the subset relation form a poset. Try \( P=\text{Poset}([[1,2],[],[1]]) \). This makes a Poset \( P \) consisting of 3 lists. You can get a nice picture (called a Hasse diagram) of this poset with the command \( P.\text{show}() \).

9. Consider the list of integers \( L=[5..10] \). Ordinary inequality defines a relation on \( L \). So \((a,b)\) is in the relation if and only if \( a \leq b \). Evaluate:

\( Q=\text{Poset}(([5..10], \lambda x, y: x<=y)) \). Then show it.

10. Can you think of another relation on the positive integers? How about “\( \geq \)”. Experiment that—and make a picture.

11. The positive integers together with the relation \( R \) where a pair \((a,b)\) is in \( R \) if and only if \( a \) divides \( b \) is a relation. So, for instance, \((1,5)\) is in \( R \) as 1 divides 5 and \((2,4)\) is in \( R \) as 2 divides 4. Here’s a Sage INTERACT that makes a nice picture (called a Hasse diagram) of the positive integers with the divisibility relation.

\begin{verbatim}
@interact
def _(n=(5..100)):
    Poset(([1..n], lambda x, y: y%x == 0)).show()
\end{verbatim}

12. Define any other Poset in Sage and make a Hasse diagram for that Poset. (Here’s one you could try: the subsets of a set form a poset. Could you code that?)
Julia Set

You can read about the Julia set at http://en.wikipedia.org/wiki/Julia_set. It is a common example of complex dynamics that can be illustrated with cool pictures.

13. First recall that the complex numbers have the form $a + bi$ where $i = \sqrt{-1}$. Let $z = 5 + 3i$. You can think of this as the vector which points from the origin to $(5, 3)$.

14. $z$ has a length (or magnitude). Find it with $\text{norm}(z)$. Use $\text{n(\_)}$ to get a numerical approximation. Similarly, every point in the plane can be viewed as a complex number, and with an associated magnitude.

15. The secret to the Julia Set INTERACT is the function $\text{complex_plot()}$ which associates colors to values of a complex function. Consider the function $f(z) = z$ where $z$ is a complex number. Make a pretty plot of this using $\text{complex_plot}(z, (-5, 5), (-5, 5))$. Remember to tell Sage that $z$ is a variable.

16. Here’s the description you get if you evaluate $\text{complex_plot}$?: “‘complex_plot’ takes a complex function of one variable, $f(z)$. The magnitude of the output is indicated by the brightness (with zero being black and infinity being white) while the argument is represented by the hue (with red being positive real, and increasing through orange, yellow, ... as the argument increases).” Now try $\text{complex_plot}(z**2, (-5, 5), (-5, 5))$. Can you explain the lines in the picture?

17. Most of the following code is defining the input numbers and various tricks for fast display of the result. The main bits of code are the definition of $f(z)$ and the call to $\text{complex_plot}$.

```python
@interact
def julia_plot(expo = slider(-10,10,0.1,2),
               iterations=slider(1,100,1,30),
               c_real = slider(-2,2,0.01,0.5),
               c_imag = slider(-2,2,0.01,0.5),
               zoom_x = range_slider(-2,2,0.01,(-1.5,1.5)),
               zoom_y = range_slider(-2,2,0.01,(-1.5,1.5))):
    var('z')
    I = CDF.gen()
    f(z) = z**expo + c_real + c_imag*I
    ff_j = fast_callable(f, vars=[z], domain=CDF)
    def julia(z):
        for i in range(iterations):
            z = ff_j(z)
        if abs(z) > 2:
            return z
        return z
    print "z <- z^{\{\} + (\{\} + \{\})*I".format(expo, c_real, c_imag)
    complex_plot(julia, zoom_x,zoom_y, plot_points=200, dpi=150).show(frame=True, aspect_ratio=1)
```
Sierpinski Gasket

You can read about the Sierpinski Gasket (or Sierpinski Triangle) at http://en.wikipedia.org/wiki/Sierpinski_triangle.

18. Try $5/2$ and $1/2$. This Sage command behaves the same way that $5/2$ and $1/2$ behaves in Python.

19. The binomial coefficient $\binom{a}{b}$ is defined to equal $\frac{a!}{b!(a-b)!}$. It equals the number of subsets of cardinality $b$ there are in a set of cardinality $a$. This number has lots of interpretations and interesting combinatorial properties. See if you can figure out $\binom{3}{2}$ by hand, and then see what you get with binomial(3, 2) in Sage.

Note that the familiar Pascal’s Triangle (see: http://en.wikipedia.org/wiki/Pascal’s_triangle) can be defined using binomial coefficients.

20. We’ve seen the range() command before but, like most things in Sage, it has powerful options that we haven’t seen (or used) before. Try range(4, 20), range(4, 20, 2), and then range(4, 20, 3).

21. The following program uses matrix_plot() which is a way to represent the pattern of a matrix by associating different colors to different entries. Try matrix_plot(matrix(2, [1, 2, 3, 4])), matrix_plot(matrix(2, [1, 2, 3, 1])), and matrix_plot(matrix(2, [1, 2, 3, 4]), cmap="hsv").

22. Define a $3 \times 3$ matrix with all different entries and use matrix_plot() to represent it.

23. Now let’s make our Gasket!

```python
def sierpinski(N):
    S=[([0]*(N//2-a//2)) +
       [binomial(a,b)%2 for b in range(a+1)] +
       ([0]*(N//2-a//2)) for a in range(0,N,2)]
    return S

@interact
def _(N=slider([2 ** a for a in range(12)],
               label="Number of iterations", default=64),
               size=slider(1, 20, label="Size", step_size=1, default=9)):
    M = sierpinski(2 * N)
    matrix_plot(M, cmap="binary").show(figsize=[size, size])```