1. Log in to your Sage/Cocalc account.
   
   (a) Start the Chrome browser.
   (b) Go to http://cocalc.com and sign in.
   (c) You should see an existing Project for our class. Click on that.
   (d) Click “New”, call it c14, then click “Sage Worksheet”.

Random Values

2. random() returns a random number in [0, 1]. Execute it a few times to see what you get.

3. Define a function my_mood() which prints “I’m happy” or “I’m sad” randomly.

   def my_mood():
       if random()<.5:
           print "I’m happy"
       else:
           print "I’m sad"

4. Use random() to define a function coin_flip() which randomly returns the string “H” (for heads) half the time and returns the string “T” (for tails) half the time. Try it a few times; your results will vary.

5. Make an interactive coin flipping program:

   @interact
def i_coins(n=input_box(5)):
       for x in [1..n]:
           print coin_flip()

Try different numbers in the box.

   It is often useful to generate random integers. It only makes sense to generate random integers from within some range of integers. We do this with randint().

6. Evaluate randint(5,100) a few times; your results will vary. This will generate random integers in the range [5, 100], including both endpoints.

7. Now try the following function. Evaluate it a few times; your results will vary!

   def sybil():
       print "My favorite number is {}!".format(randint(1,50))
Timing

For large programs or calculations that are at the edge of what’s possible. It is crucial to optimize and test the speed of your code. One simple first step is simply to time your program using Sage’s built-in `timeit()` function.

8. Here is our example of a recursive definition of the `factorial` function. The base case here is the case where the input is 0 or 1.

```python
def factorial1(n):
    if n==0 or n==1:
        return 1
    else:
        return n*factorial1(n-1)
```

Evaluate and write down what you get for `timeit("factorial1(10)")`, `timeit("factorial1(50)")`, and `timeit("factorial1(100)")`.

9. It is often intuitive to define a function recursively, but usually the same function can be defined without recursion. Here is our function `factorial2(n)` that does the same thing as `factorial(x)` but is not recursive. Now we’ll time it.

```python
def factorial2(n):
    result=1
    if n==0:
        return result
    for i in [1..n]:
        result=result*i
    return result
```

Evaluate and write down what you get for `timeit("factorial2(10)")`, `timeit("factorial2(50)")`, and `timeit("factorial2(100)")`. Compare with your previous results.

10. Now try the built-in `factorial` function. Evaluate and write down what you get for `timeit("factorial1(10)")`, `timeit("factorial1(50)")`, and `timeit("factorial1(100)")`. Compare with your previous results.

11. **Challenge.** Does `randint()` produce a *uniform distribution*? (That is, as you repeat experiments of `randint(a,b) are the number of produced outcomes of each possible integer roughly the same? Do some experiments!)