Workshop 1: Introduction to SageMath for abstract algebra

Friday, January 14, 2022

SageMath (Sage) is a free, open-source computer algebra system and programming language with many built-in mathematical functions and mathematical objects. Its syntax is based on Python, but you don't need to know Python to get started using Sage.

This workshop will teach you how to do a few things in Sage. You may want to cross-reference the "Sage" sections in the first two chapters of the online version of the textbook and possibly the Sage tutorial at `https://doc.sagemath.org/html/en/tutorial/`.

If you would like to do large-scale computations in Sage (say, as part of your final project), you should download SageMath (from `https://www.sagemath.org/download.html`). If you downloaded Sage, open a "SageMath Notebook", which is a `jupyter` notebook for Sage.

If you have not already downloaded Sage, use the Sage Cell Server (accessible at `https://sagecell.sagemath.org/`) for this workshop. The Sage Cell Server allows you to run Sage remotely from a browser window.

Now, work on these exercises and discuss them with your neighbors.

Core Exercises

(1) Try a simple command to check that the Sage kernel is working.

```
2+2
```

Press `shift+enter` to enter your command.

(2) Go to `http://abstract.ups.edu/aata/sets-sage.html` (Section 1.6 of the online textbook) and read through it if you haven't already, following the instructions and evaluating the Sage cells as you go.

(3) Here is a simple `for` loop that prints the integers i in the range $0 \leq i < 10$.

```
for i in srange(10):
    print(i)
```

Here is another `for` loop that prints the prime numbers in the range $10 \leq p < 20$.

```
for i in srange(10,20):
    if is_prime(i):
        print(i)
```

Finally, here is another `for` loop that does exactly the same thing.

```
for p in prime_range(10,20):
    print(p)
```

Evaluate each of these loops in a Sage cell. Then, write another loop to find the sum of all the prime numbers less than 100.

(4) The following is an identity for Fibonacci numbers: $\gcd(F_m, F_n) = F_{\gcd(m,n)}$. Check that this identity is true for all pairs of Fibonacci numbers up to F_{100}. The n^{th} Fibonacci number in Sage is `fibonacci(n)`, and the greatest common divisor of $a, b \in \mathbb{Z}$ is `gcd(a,b)`. (Fibonacci numbers are defined recursively by $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$.)

(5) The following example shows the syntax for defining a function as well as the syntax `a%b` for the remainder of a when dividing by b.

```
def is_even(n):
    return n%2 == 0
```

Evaluate the above in a Sage cell along with a few function calls to test that it works as expected. Then, write your own function `is_power_of_two(n)` to determine whether or not an integer is a power of 2. (You may want to use recursion.)

Bonus Exercises

(6) The following code defines a group G, the symmetric group S_3, which is a group with 6 elements that we will discuss next week. It also defines two elements of the group, s and t, in terms of something called "cycle notation", and returns a list containing the two products st and ts.

```
G=SymmetricGroup(3)
s=G([(1,2)])
t=G([(1,2,3)])
[s*t,t*s]
```

Evaluate this code, and take note that the two products are not equal to each other—the group is not abelian. By trying other products of strings of s and t, can you get enough information to write down a multiplication table for G?

(7) Write your own function to compute Fibonacci numbers in Sage, and compare its output to the `fibonacci` to check your work.

(8) Prove the identity $\gcd(F_m, F_n) = F_{\gcd(m,n)}$.

(9) Come up with a conjecture of your own about Fibonacci numbers. Investigate it numerically with Sage, and if it seems to be true, try to prove it.