Lab: 6
Getting started with Sage

Go to https://sagecell.sagemath.org/.

You can enter basic Sage commands into the textbox on the page and click "Evaluate" to get the output. To get started let's use sage like a fancy calculator. The operands for sum, difference, multiplication, and division are `+, -, *, /` .

Try the following operations:

1. Try the following operations:
   (a) `1 + 2 + 3`
   (b) `42 - 23`
   (c) `1331 * 11`
   (d) `144 / 9`

2. That's pretty standard stuff, you could more easily do those on your phone... But, can your phone calculate 200 digits of π in the blink of an eye? Try this:

   ```
   n(pi, digits=200)
   ```

3. You may have noticed that the previous question's answers were all whole numbers, but this last one was a decimal. We've been holding out on you a bit – sage does exact computations. Unless you ask it for a numerical approximation (which is what the `n( )` function was all about) it will give answers that are 100% precise – but sometimes that doesn't seem very helpful!

   Try evaluating these:

   (a) `pi`
   (b) `sqrt(2)`
   (c) `7/3`

4. Sage and its brethren are Symbolic Computer Algebra Systems. The reason it just parrots back the question to you has to do with the "Symbolic" part of that. For

example, the `sqrt(2)` thing above is regarded by Sage as a symbolic entity. It is a full and precise representation of the number we write as $\sqrt{2}$. It isn't some lame 4 or 10 or 100 digit approximation of $\sqrt{2}$. It is the real deal. Let's see what happens when we raise `sqrt(2)` to various powers (you can use either `^` or `**` for exponentiation in sage).

Try evaluating these:

(a) `sqrt(2) ^ 2`

(b) `sqrt(2) ^ 4`

(c) `sqrt(2) ^ 7`

Was the answer to that last one suprising? Or does it make sense in retrospect?

5. Much of the power of Computer Algebra comes from the same feature that makes regular Algebra useful – variables.

   There are two kinds that we need to distinguish: computer variables and mathematical variables. Computer variables are pretty easy, you can just make up whatever name you want and then start using it. For example:

```
myvar = sqrt(2) ^ 7
n(myvar)
```

   BTW, notice how the default version of the `n()` function only gives about 13 decimal places? Try this too: `n(myvar, 100)` (that's what 100 bytes of accuracy looks like – if you want a specific number of decimal places, see the example involving π up above.)

6. Mathematical variables are handled a little differently. First (other than x which is there automatically) you have to declare them to the system. The syntax looks like this: `y = var('y')`. What's happening there is that we're telling the system that `y` is a variable, but also that we want the system to print 'y' when referring to it. I think you'll get the point if you figure out what happens when you evaluate `y = var('w') ; 6*y`.

7. Let's try a little mathy something - with a couple of variables. Remember how the equation of a line is usually written as $y = mx + b$? Put the following into the sage cell:

```
y=var('y')
m = 2 ; b = 1
y=mx+b
```

The error message you get when evaluating this should look like:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [1], line 3
      1 y=var('y')
      2 m = Integer(2) ; b = Integer(1)
----> 3 y=mx+b

NameError: name 'mx' is not defined
```

To be fair, Sage usually puts out error messages that are on the cryptic side. But not this time! The message says "NameError" and it's even highlighted the thing that's wrong.

If you fix the error[1] something else a little unexpected happens. Nothing! There's no output...

This is just because the last line is an assignment – it has no return value. A very common "idiom" you'll see when looking at other people's sage code looks like this:

```
y=var('y')
m = 2 ; b = 1
y=m*x+b ; y
```

The semicolon is just a way to sneak multiple commands onto a single line. The difference here (as opposed to the code above) is that the last command isn't the assignment to the variable `y`, it's just `y` by itself - and that does have a return value, namely the contents of the variable `y`.

8. Sage has a function called `sum()` that adds things up. Next time we'll learn how use the help facility to find out how to use a Sage command, but for now, we're just going to tell you: the `sum()` function takes four arguments. The first argument is a formula for the terms of the sum, the second argument is the so-called summation variable, the last two arguments tell where the sum starts and ends.

   For example

[1] A takeaway from this is that you can't rely on implicit multiplication. If you write `7x` the system is going to think you've created a new (computer) variable whose name is "7x."

```
k=var('k')
sum(2*k+1, k, 0, 5)
```

would give us the sum of all the odd numbers from 1 to 11.

What is the sum of all the odd numbers from 1 to 201?

9. Use Sage to calculate 2^0, $2^0 + 2^1$, $2^0 + 2^1 + 2^2$, and $2^0 + 2^1 + 2^2 + 2^3$. Continue adding the next largest power of 2 until you notice a pattern in the result. What is the pattern?

10. The Sage summation facility can also do things entirely symbolically. This is sort of like the difference between computing a particular value and giving a general formula. Try the following in the Sage cell and see if you and Sage found the same pattern.

```
k, n = var('k, n')
sum(2^k, k, 0, n)
```