Last name ______________________

First name ______________________

LARSON—MATH 356–CLASSROOM WORKSHEET 01
Sage/CoCalc Introduction.

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 **s01**, 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 `#Problem 1`.
   (f) When you are finished with the worksheet, click “make pdf”, email me the pdf (at `clarson@vcu.edu`, with a header that says **Math 356 s01 worksheet attached**).

   Basics. Getting Started.

2. Evaluate `900*(1+.06*(90/365)` to find $900(1+.06(90/365))$. Click “Run” or SHIFT-ENTER to evaluate.

3. Evaluate `2**25` to find 25^2.

4. Find $550\dfrac{[1+(1.05)^{-30}]}{0.05}$

5. Evaluate `sqrt(8)` to get an *exact* expression for $\sqrt{8}$.

6. Evaluate `numerical_approx(sqrt(8))`, or simply `n(sqrt(8))` to get an *approximate* expression for $\sqrt{8}$.

7. Evaluate “pi”. Find a decimal approximation for π. Find a decimal approximation for 2π. Remember to type `2*pi`.

8. Evaluate “e”. Then use `n(e,digits=7)` to find a 7-digit approximation for e.

9. Find a 6-digit approximation for e^3.

10. Find $\log 10$. What did Sage compute? Did Sage compute the base-10 log?

11. Evaluate `plot(x**3,-2,2)` to sketch the graph of x^3 on the interval $(-2, 2)$.

12. Use Sage to sketch $\cos x$ on the interval $(-2\pi, 2\pi)$.

13. For any variable other than "x" you must tell Sage that you will use it as a variable. Evaluate `var("y")` to define "y" as a variable. Now evaluate `plot3d(x**2+y**2-2, (-1,1),(-1,1))` to sketch $g(x) = x^2 + y^2 - 2$ for $-1 \le x \le 1$ and $-1 \le y \le 1$.

14. Sage is written in Python. Type in the following program and evaluate.

```
def write_string(string_name):
    print(string_name)
```

Now type `write_string("hello world!")` and evaluate.

In order to do sophisticated calculations, or to allow for multiple inputs, you will need to define *procedures* (also called *functions*). Our "hello world!" program was the first example. It included a `print` statement. Other program features, in almost any language, include *conditional statements* (if..then..) and *loops*.

15. Type in the following procedure definition and evaluate.

```
# This function returns the absolute value of a number x
def absolute(x):
    if x>=0:
        return x
    else:
        return -x
```

16. Now test it. Evaluate `absolute(4)`, `absolute(-4)`. "#" is the *comment* symbol. Everything after "#" is ignored—and not evaluated.

```
def abs_plus_five(x):
    return absolute(x)+5
```

17. You don't have to add five, you can add *any* number by adding a *parameter*.

```
def abs_plus(x,y):
    return absolute(x)+y
```

18. Now test it. Evaluate `abs_plus(4,5)`, `abs_plus(-4,5)`, `abs_plus(-4,23)`, etc.

Boolean Expressions in Sage

A *boolean expression* is one that evaluates to True or False.

19. Evaluate `3==4`.

20. Evaluate `3==3`.

21. Evaluate `3>3`.

22. Evaluate `3>=-3`.

23. A very useful arithmetic operator in Sage is the *modulo* operator (represented by %). `a%n` gives the remainder of dividing a by n. Evaluate `5%2`. Now evaluate `6%2`. Try `99%5`.

24. Evaluate `13%2==1`.

25. Evaluate `13%2==0`.

    While "==" is used as a claim of equality of expressions (the left-hand-side and the right-hand-sides of the "==") the symbol "!=" is used to express does-not-equal.

26. Evaluate `5!=7`.

27. Evaluate `5!=5`.

28. We will *assign* a value to a variable "a". Then we will use that variable in a boolean expression. (These two lines can be typed in one cell, or each in its own cell). Type and evaluate:

    ```
    a=5
    a>2
    ```

    Boolean expressions can be combined with *boolean operators* like "and" and "or".

29. Evaluate `3==3 and 3==4`.

30. Evaluate `3==3 or 3==4`.

    Lists in Sage

    A *list* is a basic *data structure* in Python and Sage. They are represented by square brackets with comma separated numbers, strings, etc., between them (like $[2, 5, 9]$ or `["red", "blue"]`).

31. Lists can be given names. Evaluate `L=[2,5,9]`. Then evaluate `L`.

32. Lists are indexed starting with 0. Evaluate each of `L[0]`, `L[1]`, `L[2]`, and `L[3]`.

33. Lists can be combined with "+". Evaluate `[2,5,9]+[3,4,5]`.

34. Let `M=[3,4,5]`. Evaluate `L+M`.

35. If you want all the integers from x to y you can use the shorthand notation $[x..y]$. Evaluate `[3..7]`.

36. You can have a list of lists. Evaluate `L=[[0,1],[2,3],[4,5]]`. Now evaluate `L[1]`. Then evaluate `L[1][0]`. What do you think the value of `L[0][1]` is?

37. You can also use *list comprehension* to get the same behavior as `map()`. Evaluate `[abs(x) for x in [-1,2,-3]]`.