A Brief Introduction to the Sage Cloud

Leonard Carapezza

June 3, 2016

Abstract

Adapted from "A Brief Sage Tutorial" by Sarah Cobb, which was adapted from "A Brief Python Tutorial" by Mladen Bestvina.

1 Getting Started

To begin, go to **https://cloud.sagemath.com**. You can log in using a Facebook or Google account, or you can create a new account. Once you have signed in, click the button that says "create new project." Near the top of the screen you should see a box with "filename" in light grey text next to a button that says create. Enter "Tutorial" into the filename box and click the "create" button. You will be taken to a blank Sage worksheet, which is where we will do our coding.

2 Sage as a calculator

If you type 2+3 into the sage worksheet and hit enter, the cursor moves to the next line but no computation is done. To tell Sage that you want it to execute the instructions you have given, you can either click the green play button at the top or you can hold shift and then press enter. Tell Sage to compute the following inputs and see what happens:

```
7*6
7/3
7.0/3.0
7%3
range(3,7)
2**3
2**10
2^10
sqrt(2)
sqrt(2.0)
int(sqrt(2.0))
```

Notice that Sage gives you exact answers unless otherwise specified, e.g. it tells you that the square root of two is the square root of two. When you input a number as 7.0 instead of 7, this tells Sage that you want the output as a decimal.

2.1 Variables and equality

If you want to know if two quantities are equal, Sage can tell you. For example, suppose we don't know if 2 and 3 are equal, we can ask Sage to tell us by computing 2==3 (this should return False). Notice that we use two equals (=) signs to test for equality.

A single equals sign means that you are telling (not asking) Sage that one symbol is equal to another. For example, inputting x=4 tells sage that x is now equal to 4. See how this works by evaluating the following:

```
x=4
x+10
y=5
x*y
x+y==9
```

Also, using the sumbols <, >, <= and >=, Sage can tell you if one number is less than, greater than, less than or equal to, greater than or equal to another number, respectively. Finally, Sage can tell you if two numbers are not equal (which is sometimes more useful than knowing if they are equal); the symbol for this is !=.

3 Basic commands

When writing code, it is extremely helpful to be able to tell the computer how to do the following three things:

1. Repeat a set of instructions some number of times, determined in advance.
2. Repeat a set of instructions until a certain condition is met.
3. Do one of two things, depending on whether a certain condition is met or not.

These will be addressed in order.

3.1 For loop

Copy the following into the Sage worksheet and see what happens. Note that the asterices * are not part of any code but just serve to separate the examples.

```
for i in range(1,10):
    print i

***

x=5
y=15
for i in range(x,y):
    print i

***

for i in [2,17,0,1,3,5]:
    print i

***

x=5
y=3
for i in range(1,4):
    x=x+i
    y=y*i
print x
print y
```

Notice that indendation is very important as indentation tells Sage what is part of the instructions you want to repeat.

3.2 while loop

Copy the following into the Sage worksheet and see what happens.

```
x=3
while x<13:
    x=x+2
    print x
```

```
***

x=100
t=0
while x>0:
    x=x-3
    t=t+1
print t
```

3.3 if clause

Copy the following into the Sage worksheet and see what happens.

```
x=6
y=8
if x>y:
    print "x is greater than y"
else:
    print "x is not greater than y"

***

for x in range(1,10):
    if x%2==0:
       print x
```

Notice that when using 'if' you do not have to include an 'else' instruction.

4 Lists

Lists are indespensible tools for keeping track of information. Being able to create, modify and retrieve information from lists will make your life a lot easier.

Run the following commands in Sage and see what happens:

```
A=[2,4,6]
B=range(1,7)
B
A+B
A[0]
A[1]
A[2]
A[3]
A.append(8)
A
A.insert(1,8)
A
A.remove(8)
A
del A[2]
A
A[0]=100
A
A[-1]
B[2:4]
len(A)
```

4.1 List comprehension

Sage has the very nice feature that you can define a list in terms of elements that satisfy a certain condition; you don't have to give it specific elements. Copy the following examples into Sage to see this in action.

```
A=[1,2,4,8,9,10]
print [2*x for x in A]
print [2*x for x in A if x%2==0]
```

More impressively, the following one line of code produces all prime numbers between 2 and 1000.

```
[p for p in range(2,1000) if 0 not in [p%d for d in range(2,p)]]
```

5 Defining functions

Another useful feature of Sage is that you can define functions for later use. This allows one to break up a large project into smaller pieces.

For example, suppose that Sage did not know how to do multiplication and division but could only do addition and subtraction. I might want to create functions that multiply and divide numbers. This could be accomplished as follows:

```
def multiply(a,b):
    toadd=a
    while b>1:
        a=a+toadd
        b=b-1
    return a

def divide(a,b):
    quotient=0
    while a>=b:
        a=a-b
        quotient=quotient+1
    return (quotient,a)
```

Test these function by computing

```
multiply(3,6)
```

and

```
divide(122,5)
```

; you should get 18 and (24,2) respectively.

Suppose I want a function that takes in a list and removes all of the odd numbers. This could be accomplished with either of the following,

```
def removeodds(list):
    A=[]
    for i in list:
        if i%2==0:
            A.append(i)
    return A

def removeodds1(list):
    return [i for i in list if i%2!=1]
```

Try these functions on the list [1,2,3,4,6,7,8,9] to see this in action.

6 Strings

A string is a sequence of characters that Sage assumes have no additional properties. For example, if you want Sage to print the word `Hello`, you can try the command `print Hello` and you will get an error; the problem is that Sage doesn't know what H, e, l and o are supposed to be. If instead you enter the command `print 'Hello'` then Sage will print the word "Hello." The quotations tell Sage that what is inside should not be interpreted as anything more than a sequence of symbols.

Strings share many characteristics with lists, and much of the syntax is the same. Run the following commands in Sage and consider what you get for outputs.

```
print 'Hello' + 'World'

print '11' + '22'

message = 'AAABBCDE'
print message.count('A')
print message[0]
print message[5]
```

Modifying a string is not as straightforward as it is for lists. In particular, we do not have the insert, remove and delete functions for strings like we do for lists, and we cannot simply redefine an individual element of a string. However, any string can be changed into a list; run the following code to see an example:

```
message = 'ABCDEFG'
A=[i for i in message]
print A
```

Going backwards from a list to a string is easiest accomplished by starting with a blank string and adding characters one at a time

```
A=['A','B','C','D','E']
message=''
for i in A:
    message=message+i
print message
```

In my experience the easiest way to work with a string in many cases is to change it into a list, work with the list, then change back into a string. For example, if I wanted to write a function that takes a message and changes all of the 'C"s to 'D"s, I could accomplish that as follows:

```
def CtoD(message):
    listmessage=[i for i in message]
    for j in range(0,len(listmessage)):
        if listmessage[j]=='C':
            listmessage[j]='D'
    newmessage=''
    for k in listmessage:
        newmessage=newmessage+k
    return newmessage
```