Sage Basics

Jephian Lin

August 15, 2018

Python&SageMath. Python is a programming language. Because it is flexible for various purpose and easy to use, Python is a popular choice. SageMath is a open-source mathematics software built on Python and stands for "System for Algebra and Geometry Experimentation". It provides plenty of functions for mathematical computation and research.

How to install. CoCalc is an online platform for Linux, LaTeX, Python, Sage, R, etc. It is free, but you may subscribe to get better performance. You may experience the power of programming on CoCalc without the hassle of installation. Links to their websites are as below.

CoCalc Python SageMath

Registered an account on CoCalc, create a project, and create a Sage worksheet. Then embrace the wonder of Sage!

Your best friends.

shift + enter	evaluate the cell
tab	autocomplete or show the possible completions
object.	press `tab` to see functions under *object*
func?	evaluate this line to read the document of *func*
func??	evaluate this line to read the source code of *func*
Google	the answers are likly available online or in Sage Reference Manual

Assign a value.

`a = 1`	set the value of `a` as `1`

Print. Run `print a` or `print(a)` to print the value of `a`. Python 3 only accept the second syntax.

Data types.

`int`	integers, such as 2, 3, 5, ... in Sage `Integer` is more common
`str`	strings, such as `"235"`
`bool`	boolean value, namely, `True` and `False`
`tuple`	tuples, such as `(2,3,5)`
`list`	lists, such as `[2,3,5]`
`dict`	dictionaries, such as `{"two":2,"three":3,"five":5}` defined by `{key:value}`
`type(a)`	return the type of `a`

Boolean tests.

`in`	`1 in [2,3,5]` returns `False`
`not in`	`1 not in [2,3,5]` returns `False`
relation	`2==3` returns `False` options: >, >=, <, >=, and `!=` `!=` means not equal
`isinstance`	check the type `isinstance("235",str)` returns `True`

Arithmetic operators.

`+-*/`	addition, subtraction, and multiplication, division
`**` or `^`	exponent, `**` for Python, `^` for Sage `2^3` returns 8
`%`	modulus, `23%4` returns `3`
`//`	floor division, `15//4` returns `5`

Layout. Line breaks and indents are both sensitive in Python. Conventionally, an indent is four spaces. On CoCalc, I suggest go to "Account" and check the box of "Spaces instead of tabs". If you put several commands in a line, then use semi-colons `;` to separate them. Otherwise, semi-colons are optional.

The `if` statement. The following code decides the letter grade of the input `score`.

```
score = 90;
if score >= 80 and score <= 100:
    print "A";
elif score >= 70 and score <80:
    print "B";
elif score >= 60 and score <70:
    print "C";
elif score >= 0 and score <60:
    print "D"
else:
    print "Input score not valid"
```

The `for` loop. The following code prints the positive integers less than or equal to 100 that is a multiple of 5 or 7.

```
for i in range(1,101):
    if i%5==0 or i%7==0:
        print i;
```

You may use `generator` or `list`

`range(b)` return the list $0, 1, \ldots, b-1$.

`range(a,b)` return the list $a, a+1, \ldots, b-1$

`TreeIterator(n)` a generator of trees on n vertices, run the next line first

```
from sage.graphs.trees import TreeIterator
```

The `while` loop. The following code is a primitive way to find the least common multiple of 5 and 7.

```
i=1;
while True:
    if i%5==0 and i%7==0:
        print i;
        break;
    else:
        i=i+1;
```

Here `break` means to stop the loop.

Define a function. The following function will return the $\sum_{k=1}^{n} k^p$.

```
def power_sum(n,p,summand=False):
    total=0;
    for k in range(1,n+1):
        total += k^p;
        if summand:
            print k^p;
    return total;
```

Thus, `power_sum(10,1)` returns 55. The variable `summand` has a default value `False` so it is optional; when it is `True`, the function will print the summands. For example, `power_sum(10,2,True)` will print $1, 4, 9, \ldots$ and then return 385.

Call values.

`f(a,b)`	return the value of the function `f` with given inputs `a` and `b`
`L[k]`	return the value of the k-th element in the list `L`
`D[k]`	return the value that corresponding the the key `k` in the dictionary `D`

Shorthand and string formatting.
`[k^2 for k in range(6) if k%2==1]`
means `[1,9,25]`
`{k:k^2 for k in range(6) if k%2==1}`
means `{1:1,3:9,5:25]`
`n=5; print "%s+1=%s"%(n,n+1);`
prints `5+1=6`
`n=5; print "{0}+1={1}".format(n,n+1);`
prints `5+1=6`

Sublist. Suppose `a=[1,2,3,4,5]`.

`a[-2]`	returns `4`
`a[2:-2]`	returns `[2,3,4]`

Matrix. To assign

$$M = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix},$$

the following two lines do the same work.
`M=matrix([[0,1,2],[3,4,5]])`
`M=matrix(2,range(6))`
Suppose M is this given matrix.

`M[i,j]` returns the i, j-entry

`M[[0,1],[1,2]]` returns the submatrix induced on rows indexed by `[0,1]` and columns indexed by `[1,2]`.

`M[[1],:]` returns the row with index `1`

`M[:,[2]]` returns the column with index `2`

Graph. To assign $G = K_{2,3}$, each of the following three lines achieve the task, but only the first line assign the positions of the vertices. (Try `G.show()` to see the difference.)

```
G=graphs.CompleteBipartiteGraph(2,3)
G=Graph({0:[2,3,4],1:[2,3,4]})
G=Graph("D]o")
```

Here `"D]o"` is the graph6 string of $K_{2,3}$.
Use `nauty` to search graphs. The following code prints the graph6 string for all connected graphs on 4 vertices.

```
n=4;
for g in graphs.nauty_geng("%s -c"%n):
    print g.graph6_string();
```

You may check the isomorphism by first giving them a "standard" labeling and then compare their string. The following code check if `G` and `H` are isomorphic.

```
stg_G=G.canonical_label().graph6_string();
stg_H=H.canonical_label().graph6_string();
stg_G==stg_H
```