An Introduction to Sage

Giulia Mauri

Politecnico di Milano

email: *giulia.mauri@polimi.it*
website: *http://home.deib.polimi.it/gmauri*

April 15, 2015

Overview

1. First steps with SAGE

2. Number Theory

References:

[1] SAGE: http://sagemath.org
[2] PYTHON: http://docs.python.org

What is SAGE

SAGE is a free and open source mathematics package implemented using the programming language Python, but you do not need to know Python to use SAGE.

You can freely download SAGE at:

- http://sage.math.washington.edu/sage
- http://modular.fas.harvard.edu/sage
- http://echidna.maths.usyd.edu.au/sage
- http://sage.scipy.org/sage
- http://cocoa.mathematik.uni-dortmund.de/sage

Starting SAGE

After the installation, type `sage` to start the SAGE shell.
You should get:

```
> sage
--------------------------------------------------------
| SAGE Version 1.5.3, Build Date: 2007-01-05          |
| Distributed under the GNU General Public License V2.|
--------------------------------------------------------
sage:
```

To quit Sage either press Ctrl-D or type `quit` or `exit`.

You can also use for free the online Sage Cloud at:
https://cloud.sagemath.com.

Assignments, Equality and Arithmetics

Sage uses **=** for assignment.
It uses **==**, $<=$, $>=$, $<$, and $>$ for comparison.

```
sage:  a = 5
sage:  a
5
sage:  2 == 2
True
sage:  2 == 3
False
sage:  2 < 3
True
sage:  a == 5
True
```

Assignments, Equality and Arithmetics

Sage provides all of the basic mathematical operations.

```
sage: 2 ** 3 # ** means exponent
8
sage: 2 ^ 3 # ^ is a synonym for **
8
sage: 10 % 3 # for integer arguments, % means mod
1
sage: 10 / 4
5/2
sage: 10 // 4 # for integer arguments, // returns the integer quotient
2
sage: sqrt(3.4) # sqrt returns the square root
1.84390889145858
sage: exp(2)
e^2
```

Types

Sage provides variables with a type associated with it. A variable may hold values of any type within a given scope.

```
sage:  x = 1 # x is an integer
sage:  type(x)
<type 'sage.rings.integer.Integer'>
sage:  x = 1/2 # now x is a rational number
sage:  type(x)
<type 'sage.rings.rational.Rational'>
sage:  x = 'hello ' # now x is a string
sage:  type(x)
<type 'str'>
```

Functions

To define a new function in Sage, use the def command and a colon after the list of variable names.

```
sage: def is_even (n):
....:     return n%2 == 0 # blocks of code are indented
sage:  is_even (2)
True
sage:  is_even(3)
False
```

Semicolons are not needed at the ends of lines; you can put multiple statements on one line, separated by semicolons.

```
sage :  a = 5; b = 3; c = a + b;
```

If you would like a single line of code to span multiple lines, use a terminating backslash.

```
sage :  2 + \
....:     3
```

Iterating

In Sage, you count by iterating over a range of integers.

```
sage :  for i in range(3) # is like for(i=0; i<3; i++)
....:     print i
0
1
2
sage:  for i in range(2,5) # is like for(i=2;i<5;i++)
....:     print i
2
3
4
sage :  for i in range(1,6,2) # is like for(i=1;i<6;i+=2)
....:     print i
1
3
5
```

Lists

The most basic data structure in Sage is the list, which is just a list of arbitrary objects. List indexing is 0-based, as in many languages.

```
sage :  v = range(2,10)
[2, 3, 4, 5, 6, 7, 8, 9]
sage:  v[0] # 2
sage:  v[3] # 5
```

Use `len(v)` to get the length of v, use `v.append(obj)` to append a new object to the end of v, and use `del v[i]` to delete the entry of v.

```
sage :  len(v)
8
sage :  v.append(10)
sage :  del v[1]
sage :  v
[2, 4, 5, 6, 7, 8, 9, 10]
```

Lists

Use `insert(i,x)` to insert an item at a given position, use `remove(x)` to remove the first item from the list whose value is x, use `sort()` to sort the items of the list in place, and use `reverse()` to reverse the elements of the list.

```
sage :  v.insert(1,3)
[2, 3, 4, 5, 6, 7, 8, 9, 10]
sage :  v.remove(9)
[2, 3, 4, 5, 6, 7, 8, 10]
sage :  v.reverse()
[10, 8, 7, 6, 5, 4, 3, 2]
sage :  v.sort()
[2, 3, 4, 5, 6, 7, 8, 10]
```

Basic Rings

A *ring* is a mathematical construction in which there are well-behaved notions of addition and multiplication. Four commonly used rings are:

- the integers, called `ZZ` in Sage.
- the rational numbers, called `QQ` in Sage.
- the real numbers, called `RR` in Sage.
- the complex numbers, called `CC` in Sage.

```
sage :  QQ
Rational Field
```

Also the set of integers modulo n, $\mathbb{Z}_n$, is a ring. For example:

```
sage :  Zmod(26)
Ring of integers modulo 26
```

Finite Fields

Sage also knows about other rings, such as finite fields, the ring of algebraic numbers, polynomial rings, and matrix rings.
The ring of integers modulo *n* is a finite field if and only if *n* is prime. If *n* is a non-prime prime power, there exists a unique finite field GF(n) with *n* elements, which must not be confused with the ring of integers modulo *n*, although they have the same number of elements.

```
sage:gf = GF(3)
Finite Field of size 3
sage:  gf.cardinality() # return the number of elements of the finite field
3
sage:  gf.some_elements() # return a collection of elements of the finite field
[0, 1, 2]
```

Number Theoretic Functions

sage : `a = 15; b = 35; c = 28; n = 19`

sage : `gcd(a, b)` # *return the greatest common divisor of the integers a and b*

5

sage : `inverse_mod(a,n)` # *return the inverse of a modulo n*

14

sage : `factor(a)` # *return the prime factors of a*

3 * 5

sage : `divisors(c)` # *return the divisors of c*

[1, 2, 4, 7, 14, 28]

sage : `prime_divisors(n)` # *return only the prime divisors of c*

[2, 7]

sage : `euler_phi(a)` # *return the value of the ϕ function of a, that is the number of positive integers less than or equal to a that are relatively prime to a*

8

Matrices

Creation of matrices and matrix multiplication is easy and natural.

```
sage :  A = matrix([1,2,3], [3,2,1], [1,1,1])
sage :  w = vector([1,1,-4])
sage :  w * A
(0, 0, 0)
sage :  A * w
(-9, 1, -2)
```

Compute the inverse of a matrix modulus n.

```
sage :  A = matrix([13,12,35],[41,53,62],[71,68,10]% 999)
sage :  A ^(-1) % 999
[772 472 965]
[641 516 851]
[150 133 149]
```