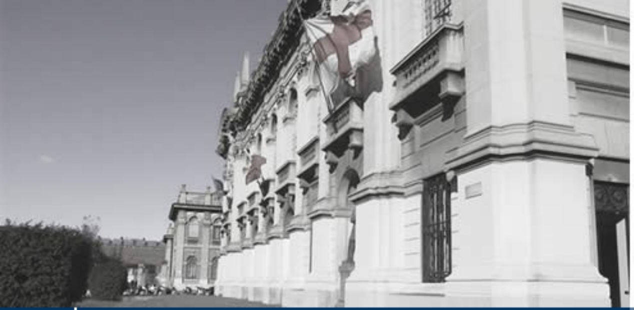

POLITECNICO DI MILANO

An introduction to SAGE

Cristina Emma Margherita Rottondi

E-mail: cristinaemma.rottondi@polimi.it; cristina.rottondi@supsi.ch

Website: http://people.idsia.ch/~rottondi/

What is SAGE?

- Sage is a free and open-source mathematics package
- Based on the mainstream programming language Python
- Headed by the mathematician William Stein, who is at the University of Washington, in Seattle
- Free download at:
  - http://www.sagemath.org

The SAGE shell

To start the SAGE shell, just type:

```
> sage
```

You should see something like:

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

To quit SAGE press CTRL-D or type `quit` or `exit`

You may also try for free the Sage Notebook at **https://cloud.sagemath.com**

Assignments

- 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
```

Basic Operations


```
sage: 2**3 # ** means exponent
8
sage: 2^3 # ^ is a synonym for ** (unlike in Python)
8
sage: 10 % 3 # for integer arguments, % means mod, i.e., remainder
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
```

Types

- Every object in SAGE has a type
- The type may be either a standard Python class or a class implemented in SAGE
- A few examples of types:

```
sage: x = 1  # x is an integer
sage: type(x)
<type 'sage.rings.integer.Integer'>
sage: y = 2/1  # x is now  a rational number
sage: type(y)
<type 'sage.rings.rational.Rational'>
sage: z = '2'  # z is a string
sage: type(z)
<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
sage: is_even(2)
True
sage: is_even(3)
False
```

- NOTE: In Python, blocks of code are not indicated by curly braces or begin and end blocks. Instead, blocks of code are indicated by indentation, which must match up exactly.
- Semicolons are not needed at the ends of lines; a line is in most cases ended by a newline. However, you can put multiple statements on one line, separated by semicolons
- If you would like a single line of code to span multiple lines, use a terminating backslash

Member Functions

- Each class of object has special member functions which apply to it:

```
sage: x.is_one()
True
sage: y.is_one()
False
```

- To determine what a function does, type (the object, '.', and) the function name, followed by a ? (and return):

```
sage: x.is_one? Type:
  builtin_function_or_method
Base Class:      <type 'builtin_function_or_method'>
String Form:     <built-in method is_one of
  sage.rings.integer.Integer object at 0x6333a20>
Namespace:       Interactive
Definition:      x.is_one(self)
Docstring: Returns "True" if the integer is 1,
  otherwise "False".
```

Iterating

- In Sage, you count by iterating over a range of integers. For example, the first line below is exactly like `for(i=0; i<3; i++)` in C++ or Java:

```
sage: for i in range(3):
          print i
```

- Output: 0,1,2
- The first line below is like for(i=2;i<5;i++)

```
sage: for i in range(2,5):
          print i
```

- Output: 2,3,4
- The third argument controls the step, so the following is like `for(i=1;i<6;i+=2)`

```
sage: for i in range(1,6,2):
          print i
```

Output: 1,3,5

Lists (I)

- The most basic data structure in Sage is the list, which is just a list of arbitrary objects. For example, the range command that we used creates a list:

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

- List indexing is 0-based

```
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: v
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
sage: del v[1]
sage: v
[2, 4, 5, 6, 7, 8, 9, 10]
```

Lists(II)

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, 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.list()  # return the 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 Euler 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]
```