Sage Tutorial

LASEC

EPFL, Switzerland

2025

Sage

- Sage is a free open source (GPL) mathematics software.
- It combines many open source packages (e.g., GP, R,. . .)
- Can be downloaded on `http://sagemath.org`.
- The reference can be found on `http://www.sagemath.org/doc/reference/`.
- More tutorials can be found online.
- Programming language: *Python 3*.
- Tutorial about Python can be found on `http://docs.python.org/3/tutorial/`

Installing

All resources on http://sagemath.org/download.html

- **Under Windows:**
  - Easiest: Install WSL https://learn.microsoft.com/en-us/windows/wsl/install and follow the instructions for Linux.

Installing

All resources on http://sagemath.org/download.html

- **Under Windows (Alternative):**
  - Alternative: https://learn.microsoft.com/en-us/windows/wsl/install
    1) Download and install the lastest version of VirtualBox https://www.virtualbox.org/wiki/Downloads
    2) Download .ova file of the lastest version of Sage http://sage.mirror.garr.it/mirrors/sage/win/index.html
    3) Run downloaded .ova file and Click Import
       * Recommended use: Run Sage-VM, open browser on the **host** OS, go to http://localhost:8000

Installing

All resources on http://sagemath.org/download.html

- **Under macOS:**
  - Install Homebrew from https://brew.sh and run `brew install sage`
  - Use pre-built binaries:
    1) Download the dmg (from http://sage.mirror.garr.it/mirrors/sage/osx/index.html), double click on it.
    2) Drag the sage folder to e.g. /Applications
    3) Use finder to visit the sage folder, double click the ”sage” icon.
    4) Select to run it with ”Terminal”
    5) SAGE should pop up in a window.
    6) For the graphical notebook, type notebook() and open the URL http://localhost:8000 in a browser (on a single user machine, using inotebook() is a little easier)
  - Build from sources (instructions on http://doc.sagemath.org/pdf/en/installation/installation.pdf)

Installing

All resources on http://sagemath.org/download.html

- **Under Linux:**
  - On GNU/Linux Debian version ≥ 9, Ubuntu version ≥ 18.04, Arch Linux...: binaries (`sagemath`), browser interface (`sagemath-jupyter`) and documentation (`sagemath-doc` on Arch, else `sagemath-doc-en`).”
  - Manually install pre-built binaries (instructions on http://doc.sagemath.org/pdf/en/installation/installation.pdf)
  - Build from sources (instructions on https://doc.sagemath.org/pdf/en/installation/installation.pdf)
  - Use the VM (but why would you. . .)

Ways of Using Sage

- Command line interpreter.
  $\sim$ sage
- Write a compiled program or a Python script.
  $\sim$ sage file.sage
- Use the Notebook graphical interface.

The Interpreter

- Best way of testing commands.
- Autocompletion with *Tab*.
- Information about command if you append *?* to a command.
- Information + source code if you append *??* to the command.

The Notebook Interface

- If Sage is sufficiently old, start it from command line by typing notebook().
- You can also run sage with Jupyter notebook, via sage –notebook jupyter.
- Easy way to display results and remember commands.

Some Basic Python

- Indentation is vital.
- A function is defined in the following way:

```
def fun(param1):
    param2 = param1+2
    #Comments
    return param2
```

- True, False (beware of the capital letter)

Some Basic Python

- Conditional statements:

```
if x < 0 or x > 2:
    # Do something
elif x > 1 and not y>0 :
    #Do something else
else :
    #...
```

- Ranges:

```
for x in range(10)
```

Some Basic Python

- While statement:

```
x = 0
while x < 10 :
    print(x)
    x += 1
```

Some Basic Python

- Condensed way of defining a list:

```
>>> a = [x^2 for x in range(10) if x%2 == 0]
>>> a
[0, 4, 16, 36, 64]
```

- Some limited functional programming (also check **reduce**)

```
>>> map(lambda x:x%16==0, a)
[True, False, True, False, True]
```

- OO programming (not covered in this tutorial)

Printing

```
>>> a = 10
>>> b = 12
>>> L = [1 , 2 , 3]
>>> print("a is", a, "and b is", b)
a is 10 and b is 12
>>> print("L=",str(L))
L= [1, 2, 3]
```

How to get a floating point value

- Exact value will be preferred. If you need a value in $\mathbb{R}$, cast it into Real field *RR*. You can set the precision to an chosen value arbitrarily big.

```
sage: sqrt(237)
sqrt(237)
sage: RR(sqrt(237))
15.3948043183407
```

Rings, Fields, ...

- RR : Real field
- QQ : Rational field
- ZZ : Ring of integers
- Integers(234) or Zmod(234) : Ring of integers modulo 234
- Polynomial Ring

```
sage: P.<z> = GF(2)[]
sage: P
Univariate Polynomial Ring in z over Finite
    Field of size 2 (using NTL)
```

Finite Fields

- ```
  sage: F = GF(5)
  sage: F
  Finite Field of size 5
  ```

- ```
  sage: F.<x> = GF(2^6)
  sage: F
  Finite Field in x of size 2^6
  ```

Primes

- Check if an integer is prime :

```
sage: is_prime(23457)
False
```

- Getting the next prime :

```
sage: next_prime(23457)
23459
```

Modular Computations

```
sage: a = Mod(12,15)
sage: b = Mod(10,15)
sage: a+b
7
sage: a = Mod(12,15)
sage: b = Mod(10,16)
sage: a+b
TypeError: unsupported operand parent(s) for '+': '
    Ring of integers modulo 15' and 'Ring of integers
     modulo 16'
```

Modular Exponentiation

Compute $x^e \bmod n$:

```
sage: n = 2345
sage: a = 123456578901234567890
sage: x = 12354
sage: Mod(x^a,2345)
RuntimeError: exponent must be at most
    9223372036854775807
sage: power_mod(x,a,n)
841
sage: u = Mod(x,n)
sage: u^a
841
```

In the first case, the result is an integer in $\mathbb{Z}$. In the second case, the result is in the ring of integers mod n.

How to Get the Integer?

```
sage: a = Mod(12,23)
sage: type(a)
<type 'sage.rings.finite_rings.integer_mod.
    IntegerMod_int'>
sage: b = a.lift()
sage: type(b)
<type 'sage.rings.integer.Integer'>
sage: b = lift(a)
sage: type(b)
<type 'sage.rings.integer.Integer'>
```

Careful with Your Types

This can be a big source of pain if forgotten!

```
sage: a = Mod(12,23)
sage: type(a)
sage: b = a.lift()
sage: chr( b + ord('a') )
'm'
sage: chr( a + ord('a') )
'\x11'
```

Extended Euclidean Algorithm

- For gcd:

```
sage: gcd(12,17)
1
```

- For Extended euclidean algorithm:

```
sage: xgcd(12,17)
(1, -7, 5)
```

Making a plot

- Plotting a function:

```
plot(lambda x:x+2, 0,10)
```

- Plotting values:

```
sage: x = range(10)
sage: y = map(lambda u : sqrt(u), x)
sage: list_plot(zip(x,y))
```

Other Useful Functions

- Euler's Totient Function

```
sage: euler_phi(24)
8
```

- Factoring (can be very slow):

```
sage: a = 123456789123456789123456789
sage: a.factor()
3^3 * 757 * 3607 * 3803 * 440334654777631
sage: list(a.factor())
[(3, 3), (757, 1), (3607, 1), (3803, 1),
   (440334654777631, 1)]
sage: a.factor(limit=3000)
3^3 * 757 * 604025584047442580965 1
```

Other Useful Functions 2

- All words over an alphabet

```
sage: Words(4,5)
Words of length 5 over {1, 2, 3, 4}
sage: for w in Words([5,6,7],2):
....:     print(str(w))
....:
55
56
57
65
66
67
75
76
77
```

Remarks

- Try using virtual environments to isolate Sage from other Python environments.
- Do not try to install via pip (there are no official packages).
- More tutorials can be found online.
- You can run your sage code on `cocalc.com`