

# Python Lecture

## Day 1

Dr. Giuliano Taffoni – Dr. Giuseppe Murante

# Programming in science

- Get data (simulation, experiment control),
- Manipulate and process data,
- Visualize results (to understand what we are doing!),
- Communicate results: produce figures for reports or publications, write presentations.

# Programming Language types

- A programmer write a source code using a specific programming language defined by a specific syntax;
- **Compiled Languages:** the source code is transformed (and optimized) by a compiler into machine code to be executed by the processor.

```
void store(double *a, double *b, double *c)
{
    *c = *a + *b;
}
```



```
.text
.p2align 4,,15
.globl store
.type store, @function
store:
movsd (%rdi), %xmm0 # Load *a to
%xmm0
```

# Interpreted Language

- **Interpreted Language:** an interpreter converts the source code into machine code each time you run the program, one line at a time. It starts interpreting each instruction immediately upon execution;
- **Semi-interpreted:** the source code is transformed into an intermediate format (bytecode) which is then interpreted at runtime.

Myprogram.py  Myprogram.pyc

# Programming Languages

- Compiled languages: C, C++, Fortran, etc.
  - Very fast. Very optimized compilers. For heavy computations, it's difficult to outperform these languages.
  - Some very optimized scientific libraries have been written for these languages. Example: BLAS (vector/matrix operations)
- Drawbacks:
  - Painful usage: no interactivity during development, mandatory compilation steps, verbose syntax (&, ::, `}}`, ; etc.), manual memory management (tricky in C). These are difficult languages for non computer scientists.

# Scripting languages

- Scripting languages: Matlab, Scilab, Octave, Igor, R, IDL, etc.
  - Very rich collection of libraries with numerous algorithms, for many different domains. Fast execution because these libraries are often written in a compiled language.
- Not Pleasant development environment: comprehensive and well organized help, integrated editor, etc.
- Open-source, free, or not
  - Some features can be very advanced (statistics in R, figures in Igor, etc.)
  - Many available algorithms
  - Some software are dedicated to one domain. Ex: Gnuplot or xmgrace to draw curves. These programs are very powerful, but they are restricted to a single type of usage, such as plotting.

# What is Python?

- Programming and scripting language
- “Object Oriented”
- Procedural programming and functions → we can decide our programming style
- Version: 2.x and 3.x
  - 3.x is **not** backwards compatible with 2.x

# Main characteristics

- Interpreted language (or semi–interpreted)
- Interactive
- Dynamic typing
- Automatic memory management

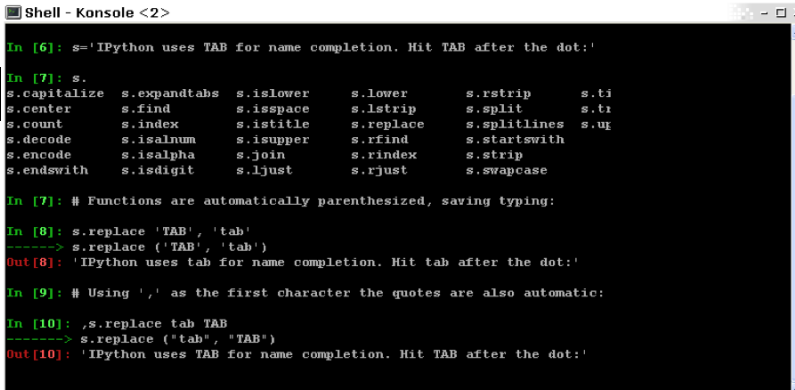


# Main characteristics

- A very readable language with clear non-verbose syntax
- Multi-platform (Unix/Linux, Mac OS X, Windows, Android)
- Large variety of high-quality packages are available for various applications, from web frameworks to scientific computing.
  - E.g. numpy, matplotlib, astropy etc
- Open source

# Scientific Python building blocks

- Python, a generic and modern computing language
  - Python language: data types (string, int), flow control, data collections (lists, dictionaries), patterns, etc.
  - Modules of the standard library.
  - A large number of specialized modules or applications written in Python: web protocols, web framework, etc. ... and scientific computing.
  - Development tools (automatic testing, documentation generation)
- IPython, an advanced Python shell



```
Shell - Konsole <2>

In [6]: s='IPython uses TAB for name completion. Hit TAB after the dot:'

In [7]: s.
s.capitalize s.expandtabs s.islower s.lower s.rstrip s.ti
s.center s.find s.isspace s.lstrip s.split s.tr
s.count s.index s.istitle s.replace s.splitlines s.uj
s.decode s.isalnum s.isupper s.rfind s.startswith s.ug
s.encode s.isalpha s.join s.rindex s.strip
s.endswith s.isdigit s.ljust s.rjust s.swapcase

In [7]: # Functions are automatically parenthesized, saving typing:

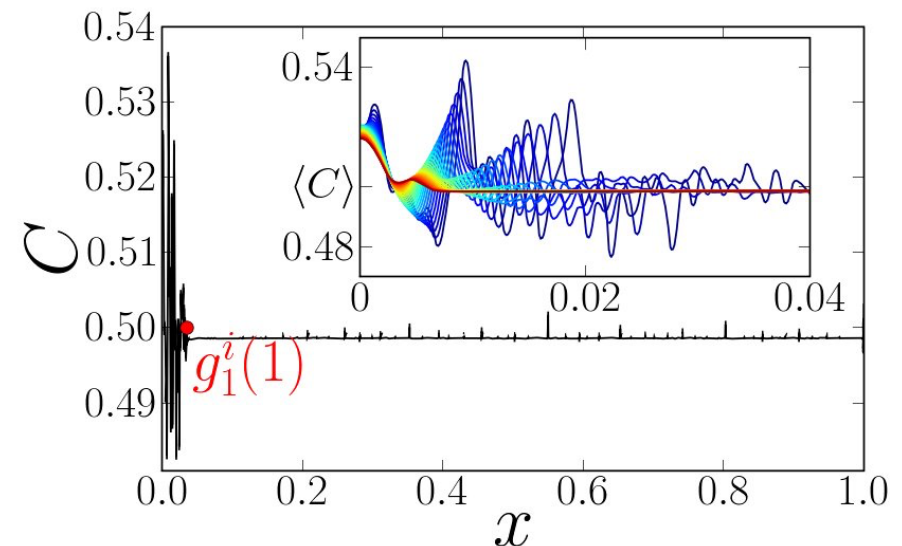
In [8]: s.replace 'TAB', 'tab'
-----> s.replace ('TAB', 'tab')
Out[8]: 'IPython uses tab for name completion. Hit tab after the dot:'

In [9]: # Using ',' as the first character the quotes are also automatic:

In [10]: ,s.replace tab TAB
-----> s.replace ('tab', 'TAB')
Out[10]: 'IPython uses TAB for name completion. Hit TAB after the dot:'
```

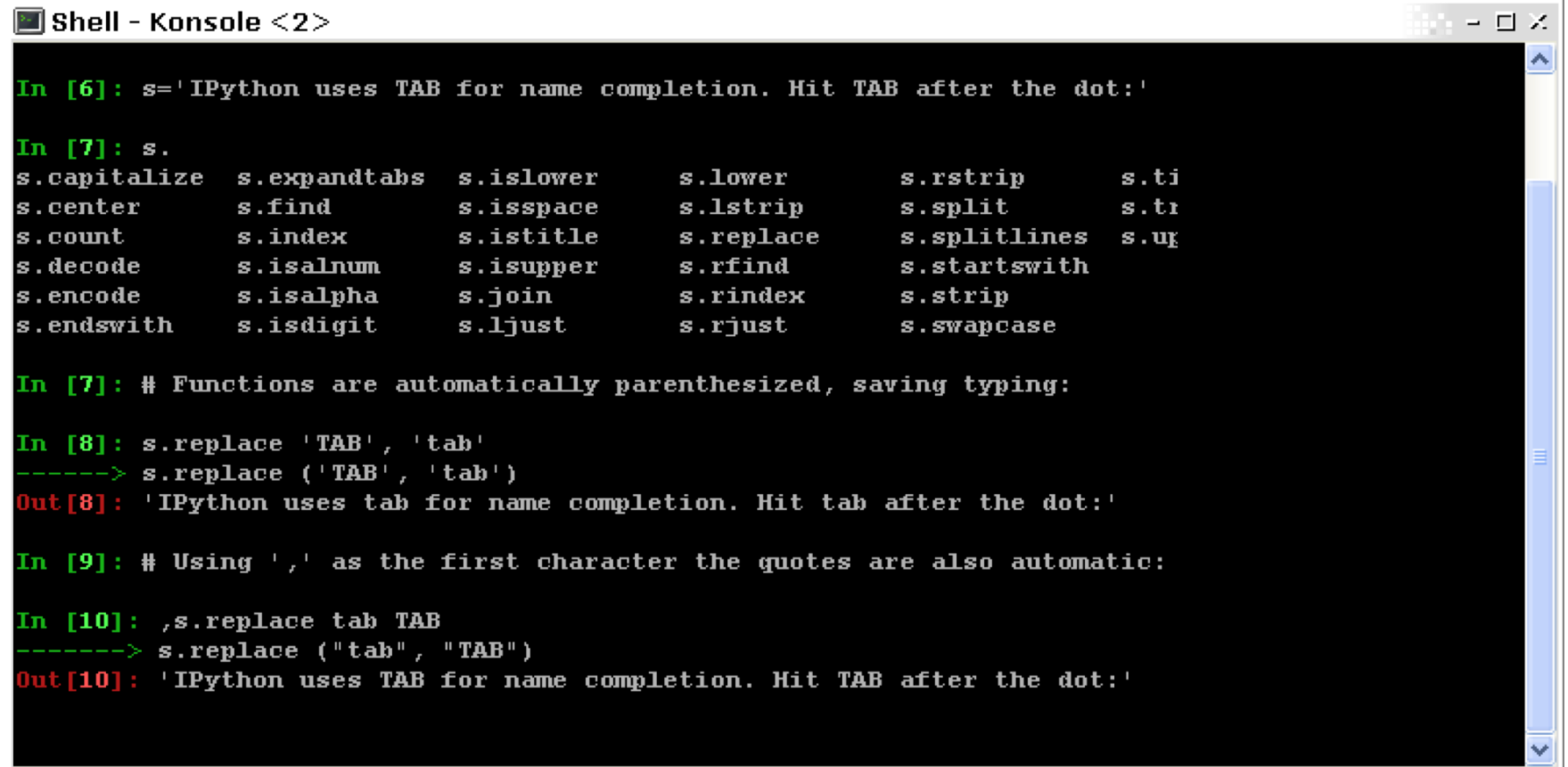
# Scientific Python building blocks

- Numpy : provides powerful numerical arrays objects, and routines to manipulate them.
- Scipy : high-level data processing routines. Optimization, regression, interpolation, etc
- Matplotlib : 2-D visualization, “publication-ready” plots
- Mayavi : 3-D visualization



# Interactive Python

- ipython



```
Shell - Konsole <2>

In [6]: s='IPython uses TAB for name completion. Hit TAB after the dot:'

In [7]: s.
s.capitalize s.expandtabs s.islower s.lower s.rstrip s.ti
s.center s.find s.isspace s.lstrip s.split s.tr
s.count s.index s.istitle s.replace s.splitlines s.uy
s.decode s.isalnum s.isupper s.rfind s.startswith
s.encode s.isalpha s.join s.rindex s.strip
s.endswith s.isdigit s.ljust s.rjust s.swapcase

In [7]: # Functions are automatically parenthesized, saving typing:

In [8]: s.replace 'TAB', 'tab'
-----> s.replace ('TAB', 'tab')
Out[8]: 'IPython uses tab for name completion. Hit tab after the dot:'

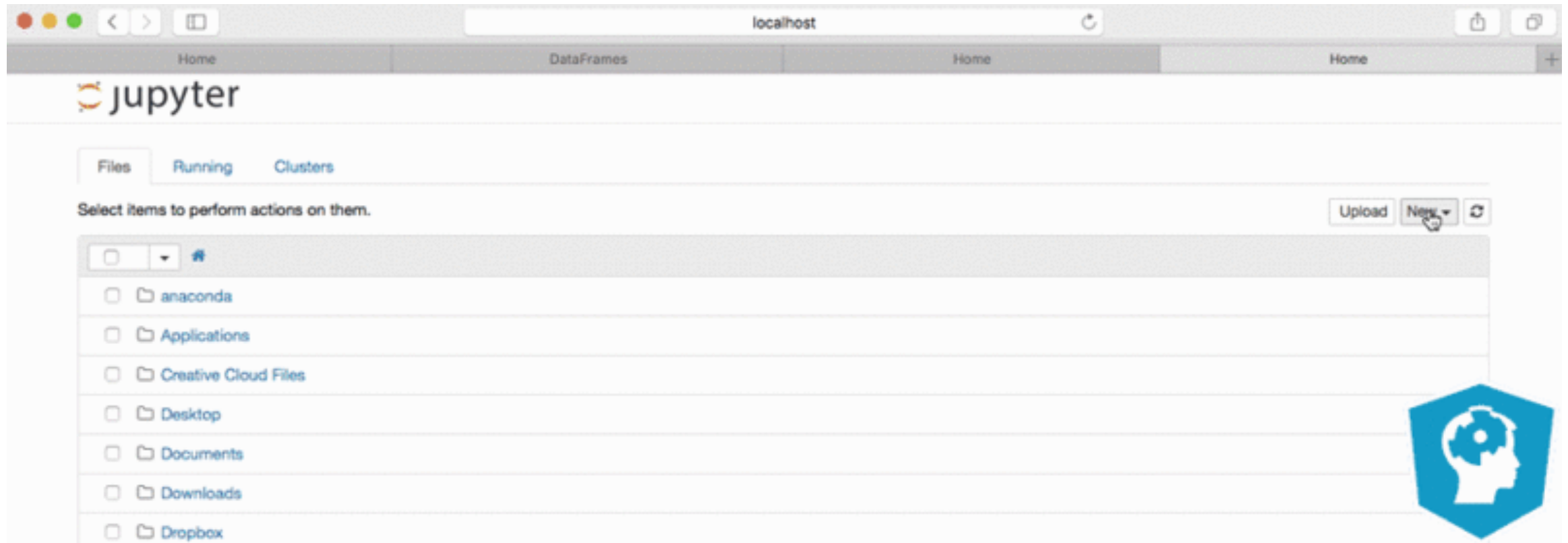
In [9]: # Using ',' as the first character the quotes are also automatic:

In [10]: ,s.replace tab TAB
-----> s.replace ("tab", "TAB")
Out[10]: 'IPython uses TAB for name completion. Hit TAB after the dot:'
```

# Jupyter

- Jupyter" is a loose acronym meaning Julia, Python, and R.
- a web application in which you can create and share documents that contain live code, equations, visualizations as well as text,
- A notebook" or "notebook documents" denote documents that contain both code and rich text elements, such as figures, links, equations,
- As a server–client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser.

# Working with Jupyter



# Variables

- Programming is based on data (variables) and instructions (code and functions). [...] Instructions tell the computer what to do with variables.
- “We store values in variables. Each variable is identified by a *variable name*. [...] Each variable has a *variable type*”
- ”a variable must be defined in a declaration statement”
  - Serve a definire il nome e il tipo
- Variables are given a value through the use of an *assignment statement*.

```
La_mia_variabile = il_suo_valore
```

# Variables in Python

- All the elements are objects: variables, functions, class, etc.
- It is a typed language: objects are typed
- Dynamic Language: objects are not declared since the beginning of the code but only when used (assignment statement)



# Comments e documentation

```
'''
```

```
Commento su piu' righe  
Utile per documentare un programma  
'''
```

```
# commento su riga singola
```

# Some examples

a=3

- Create object “number” with value 3 and assign name a

p=[1, 2 , 3 , 4 ]

- Create object list with 4 elements and assign name p

k=p

- Assign a new name to list p

a="tre "

- Create object string and assign name a. Object 3 is destroyed

# Numeric types

`int, long, float, complex`

- `a=3`
- `b=3.1415926`
- `c=15000L`
- `d=a+b`
- `e=7/2`
- `f=float(333)`
- `g=complex(1.5,2)`
- `h=None`
- `i=True`
- `print f * g`
- `print e`

# Operations

- $x + y$
- $x - y$
- $x * y$
- $x / y$
- $x // y$
- `long(x)`
- `float(x)`
- `complex(re,im)`
- `c.conjugate()`
- `divmod(x, y)`
- `pow(x, y)`  $x ** y$
- `abs(x)`
- `int(x)`

# Assignment operators

Operator	Example	Equivalent to
<code>+=</code>	<code>a += 5</code>	<code>a = a + 5</code>
<code>-=</code>	<code>a -= 5</code>	<code>a = a - 5</code>
<code>*=</code>	<code>a *= 5</code>	<code>a = a * 5</code>
<code>/=</code>	<code>a /= 5</code>	<code>a = a / 5</code>
<code>%=</code>	<code>a %= 5</code>	<code>a = a % 5</code>

# Excercise

- create a file *my\_file.py* in a code editor, and add the following lines:

```
s = 'Hello world'  
print(s)
```

- Run in ipython

```
In [1]: %run my_file.py
```

- And from shell

```
Python my_file.py
```

# Bit Operations

$x \mid y$

$x \wedge y$

$x \& y$

$x \ll n$

$x \gg n$

$\sim x$

```
>>> x=3
```

```
>>> y=10
```

```
>>> print bin(x), bin(y)
```

```
>>> print bin(x|y)
```

```
>>> print bin(x^y)
```

```
>>> print bin(x&y)
```

```
>>> print bin(x<<3)
```

```
>>> print bin(x>>4)
```

```
>>> print bin(~x)
```

# Strings

- Immutable objects that define a set of characters.

A="Corso di Laboratorio di Astronomia"

B='Corso di Cosmologia Teorica'

- String can be concatenated

C = A + B

- I can access to single char or to a "sub-string" (**slice**), but I cannot modify them

A[6]

A[3:7]

A[:5]

A[-1]

~~A[5]='k'~~



# Check that it is a typed language

- Python make a distinction between string and integers even if we do not declare them.

```
A="Corso di Astronomia"
```

```
C = A + 7
```

```
TypeError: cannot concatenate 'str' and 'int' objects
```

```
type(A) → str
```

# Containers

- String “...” , List [...], Dictionary and Tuple (...)
- List and Tuples are non homogeneous collections of objects (int, str, char, objects etc)
- Lists are mutable, tuple are immutable objects

```
a='corso di Astronomia'  
b=['a',3,5,3.14]  
c=(3,'x',0,'I')
```

```
3 in b  
'h' not in a  
b + b  
b + c  
a * 3, 4 * c  
a[4]  
del(b[3])  
len(a) min(b) max(b)
```

# Lists Operations (but not tuple!)

- `lista.reverse()`
- `Lista.append(cmp)`
- `lista.sort([cmp[,key[,reverse]])]`
- `map(fun, lista)`
- `filter(fun, lista)`
- `reduce(fun, lista)`



Argomenti opzionali

`fun` == Funzione generica

# Excercise

```
In [80]: import operator as op
In [81]: op.mul?
Docstring: mul(a, b) -- Same as a * b.
Type:      builtin_function_or_method
In [82]:
```

How can I sum all the elements of a list?

How can I multiply all the element of a list?

# Lists: let slice them

List[start:stop:stride]

A[0:3] (A[:3])

B=range(1,10)

B[2:8:2]

B[-3:]

	0	1	2	3	4	5	
	-6	-5	-4	-3	-2	-1	
<b>a:</b>	12	14	22	27	31	44	
	0	1	2	3	4	5	6
	-6	-5	-4	-3	-2	-1	

# Exercise

- Use slice operations to reverse a list

`['red', 'blue', 'green', 'black', 'white']`

# Dictionary: array associativi

- A dictionary is basically an efficient table that maps keys to values. It is an unordered container separated by ','

```
>>> HEADER = {'Name': 'M31', 'RA': 121.1743, 'DEC': -21.5733, 'REDSHIFT': -0.001001,
'OBSERVER': taffoni, 'COMMENT': new}
>>> HEADER.keys()
>>> HEADER.values()
>>> HEADER['Name']='NGC224'
```

- Some methods

```
dict.has_key('c')
dict.keys()
dict.values()
dict.copy() [shallow copy]
```

# Copy of objects and references

- When I make an assignment **a=b** non I do not copy the value but I duplicate the reference
- It exists shallow copies and deep copies.

```
>>> M = range(10)
>>> C = M
>>> C[5]=56
>>> print M
>>> B = M[:]
>>> id(B)
```

- `copy.copy()` shallow copy
- `copy.deepcopy()` deepcopy



# List comprehension

- It is a “mathematical” approach to list creations (extention of `map`)

Math style

```
S = {x2 : x in {0 ... 9}}  
V = (1, 2, 4, 8, ..., 212)  
M = {x | x in S and x even}
```

- In python

```
>>> S = [x**2 for x in range(10)]  
>>> V = [2**i for i in range(13)]  
>>> M = [x for x in S if x % 2 == 0]
```

# Some examples

- numbers

```
>>> N = [j for i in range(2, 8) for j in range(i*2, 50, i)]  
>>> M = [x for x in range(2, 50) if x not in N]  
>>> print M  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

- But not only numbers



```
>>> words = 'The quick brown fox jumps over the lazy dog'.split()  
>>> stuff = [[w.upper(), w.lower(), len(w)] for w in words]
```

# Modules

- A module is a file containing Python definitions and statements.
- You can organize your program in more than one script
- Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter.
- Such a file is called a *module*; definitions from a module can be *imported* into other modules or into the *main* module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

# Using Modules

- When the `import mymodule` statement is executed, the module `mymodule` is searched in a given list of directories (PYTHONPATH)

`PYTHONPATH=$PYTHONPATH:/home/emma/user_defined_modules`

- Modify the `sys.path` variable itself within a Python script

```
import sys
new_path = '/home/emma/user_defined_modules'
if new_path not in sys.path:
    sys.path.append(new_path)
```

# Scientific Modules

- Numpy
- Scipy
- Matplotlib
  
- Make your own one...

# References

- <https://www.scipy-lectures.org/index.html>
- <http://docs.python-guide.org/en/latest/>

# Next on this channel

- Write a program in python
- Functions
- Read arguments from command line
- Python 4 science: numpy e scipy
- Read FITS files
- Plots
- Some data reduction examples