Introduction to `python`

Modules, Notebooks and `pip`

Mauricio Sevilla

email= j.sevillam@uniandes.edu.co

11.03.19

On the rest of the course, we will use the `jupyter notebook` s to do our codes.

During this class we will learn how to use the notebook, install packages (We will need this further), and importing modules constructed by ourselves.

`pip`

While programming, we will need to have libraries/packages, these Libraries are code written by someone else that we can use without program it.

The construction of these libraries/packages is done by writing functions so that we do not care about how the implementation is exactly done, but by the input and output, that is why you have to read the documentation!!.

To use the libraries/packages, they must be installed, and there are several ways to do this.

For example, let us suppose that we don't have `python3` on our computers, to install it we use

- *on `Ubuntu`*

```
sudo apt-get install pyt
hon3
```

Or, on the newest version

```
sudo apt install python3
```

- *on `MacOS`: There are several ways to install it, for instance using `macports` or `homebrew`*

  - *`macports`*

    ```
    sudo port install python37
    ```

  - *`homebrew`*

    ```
    brew install python3
    ```

During the rest of the slides, we will use `homebrew` to show the installation. More information: `macports` (https://www.macports.org), `homebrew` (https://brew.sh/).

Once we have `python3` we would like to have some packages to use them on our codes, for example `numpy` offers a very useful data type implemented `numpy.array`, and if we want to use them, we would have to install it.

The simplest way to do this is,

- *on `Ubuntu`*

```
sudo apt install python3-n
umpy
```

- *on `MacOS`*

```
brew install python3-numpy
```

It is really easy, but sometimes different versions of libraries have implemented new routines, and the task of compatibility and update among them can get complicated.

To solve this, `pip` was created. `pip` is a `python` package installer. doc (https://pypi.org/project/pip/)

First of all, we have to install it,

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

and then run

```
python3 get-pip.py
```

The installing documentation can be found here link (https://pip.pypa.io/en/stable/installing/)

Then we install our packages just by using

```
pip3 install numpy
```

During the course, we are also going to use `scipy` and `matplotlib`.

Jupyter Notebook

> *The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.* *link (https://jupyter.org)*

The recommended installaton is link (https://jupyter.org/install)

```
python3 -m pip install --upgrade pip
python3 -m pip install jupyter
```

Once installed, type on the terminal prompt

```
jupyter notebook
```

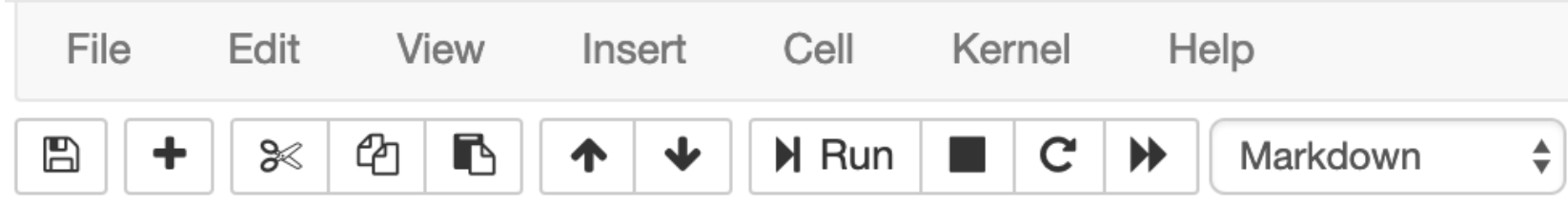
Why notebooks?

The `jupyter notbook` makes very easy to document our code, because allows us to have rich markdown text while programming.

For example, these slides are made on a `jupyter notebook` .

Another advantage is that `jupyter notebook` s supports different languages (kernels) See link (https://github.com/jupyter/jupyter/wiki/Jupyter-kernels).

Here you can see the options yuo have on the notebook

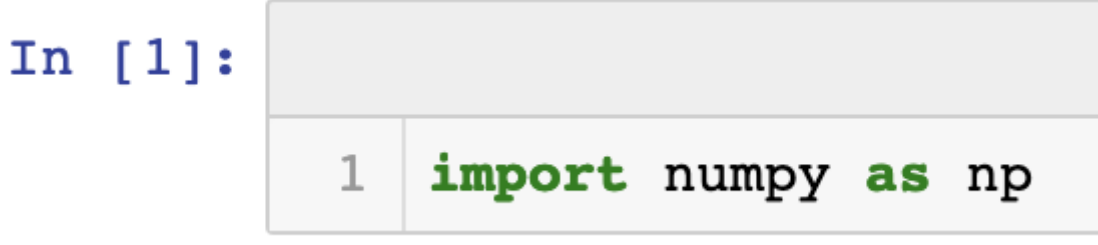


Here you can find,

- Save
- New cell
- Cut, copy and paste.
- Move cell
- Kernel options: run, stop, restart, restart and run all.
- Cell type: Markdown and code among others.

To execute a cell (markdown or code), `Shit+enter`.

On the code cells, you will see one small number `In[1]`



That number tells you the order of execution of the cells.

> *Note You can erase an already runned cell, but their effects will remain!*

If you see `In[*]` means that is running or waiting.

Let's see some examples.

- Here you can find an excelent material for the markdown cells link1 (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet), link2 (https://www.markdownguide.org/basic-syntax/).

Modules

One can construct the libraries/package ourself, these are called `Modules` , let us construct a file with some functions on it, for example

In [20]:
```
%%bash
more prog.py
```

```
def print_message(string):
    print(string)
def sum_nums(a,b):
    return a+b
def fac(n):
    if n==0 or n==1:
        return 1
    else:
        return n*fac(n-1)
```

There you can see how to use *magic cells* on the notebook. I used a bash line to print a file on the notebook!

The file `prog.py` have implemented some functions on it, the idea is that we want to use them on a different file.

first, if we have the code on the same folder, we just import the functions by using

In [26]:
```
import prog
```

We use `prog` because is the name of the file without extension.

if we do not know what the functions names are, we can use the function `dir`

In [37]:
```
dir(prog)
```

Out[37]:
```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'fac',
 'print_message',
 'sum_nums']
```

So, now we know that the functions `fac` , `print_message` , `sum_nums` are implemented, so let us use them.

the basic syntax is `library.functions(parameters)`, for instance

```
In [28]: prog.print_message('test')
```

```
test
```

```
In [29]: prog.sum_nums(1,2)
```

```
Out[29]: 3
```

```
In [30]: prog.fac(6)
```

```
Out[30]: 720
```

You can also use a *nickname* for the library while programming

In [7]:
```
import prog as p
```

In [8]:
```
p.fac(3)
```

Out[8]: 6

In [31]:
```
p.print_message('Test 2')
```

```
Test 2
```

This is also used but NOT RECOMMENDED!, because there are some libreries with different functions but with the same name.

In [10]:
```
from prog import *
```

In [11]:
```
print_message('Test 3')
```

```
Test 3
```

In [12]:
```
fac(4)
```

Out[12]:
```
24
```

In [38]:
```
fac(5)
```

Out[38]:
```
120
```

You can also import a single function

In [39]:
```
from prog import fac
```

In [40]:
```
fac(10)
```

Out[40]:
```
3628800
```