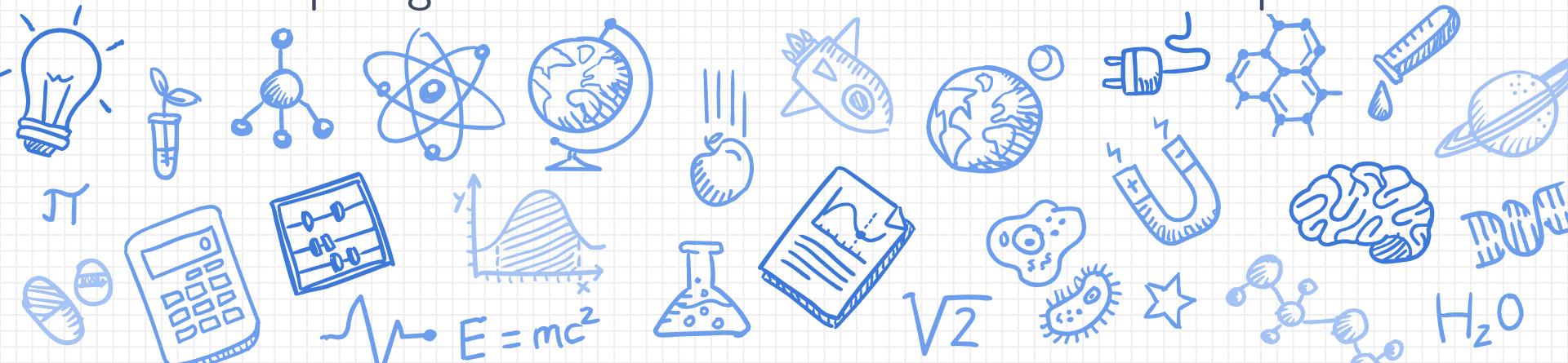


# Tools for Python

June Sallou, [june.benvegnu-sallou@irisa.fr](mailto:june.benvegnu-sallou@irisa.fr)

<https://github.com/Jnsll/ModelisationScientifique>



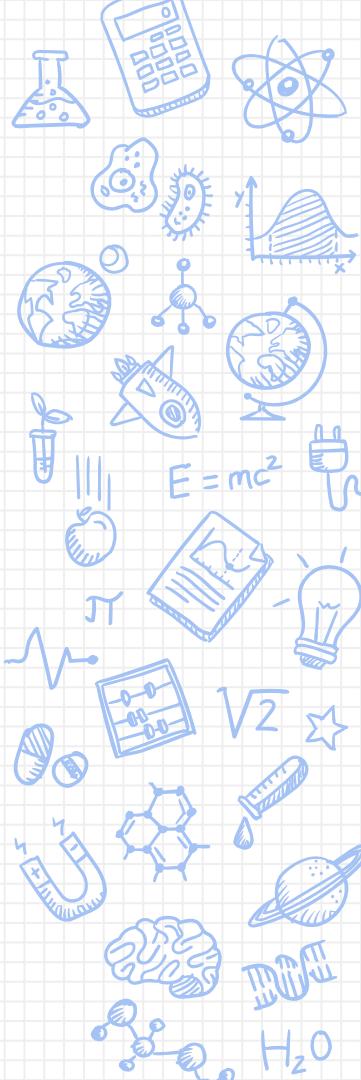
# Modules / Libraries

# What is a Module / Library ?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

To create a module just save the code you want in a file with the file extension .py.

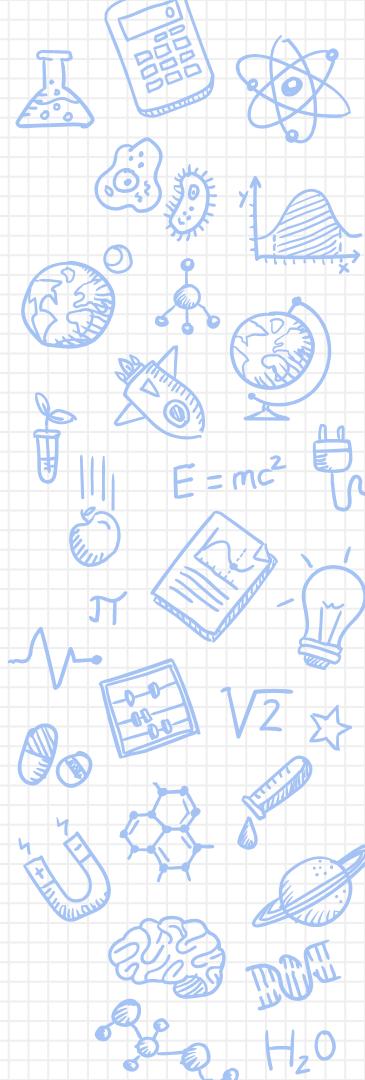


# What is a Module / Library ?

# MyModule.py

```
def greeting(name):
```

```
print("Hello, " + name)
```

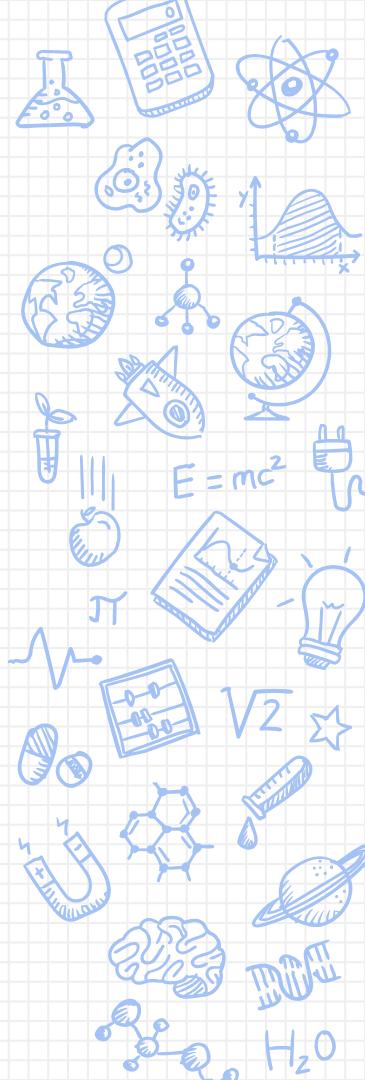


# What is a Module / Library ?

---

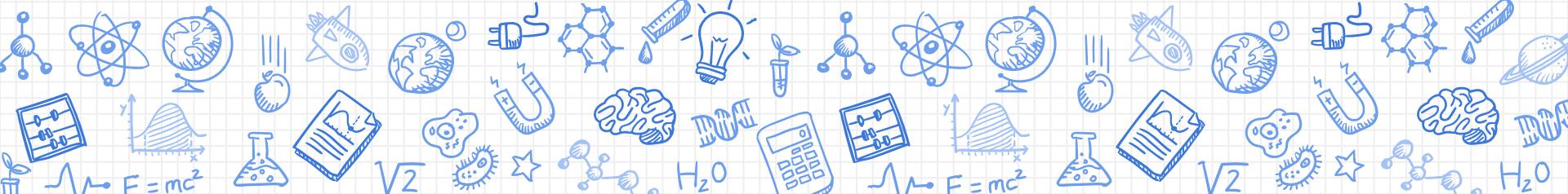
```
>>import mymodule
```

```
>>mymodule.greeting("Jonathan")
```



# Managing packages / libraries

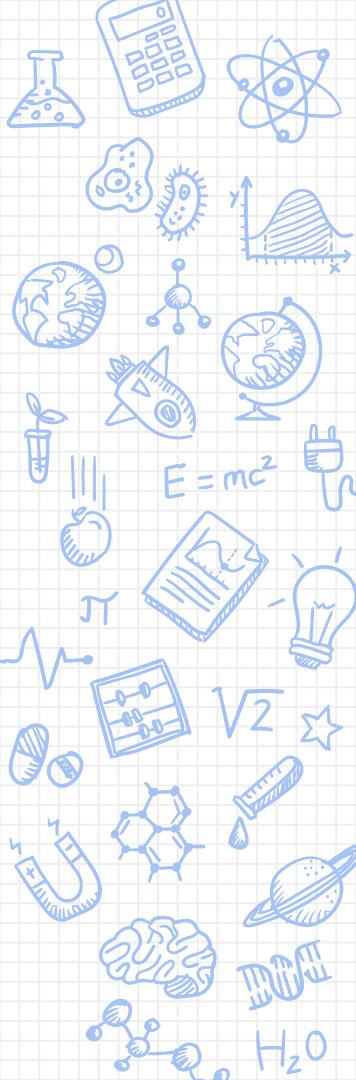
# Pip



# Pip

---

- ✗ Installs packages from the Python Package Index (PyPI) which is a repository of software for the Python programming language.  
<https://pypi.org/>
- ✗ has a number of subcommands: “search”, “install”, “uninstall”, “freeze”, etc
- ✗ lack of synchronisation (with requirements.txt file)

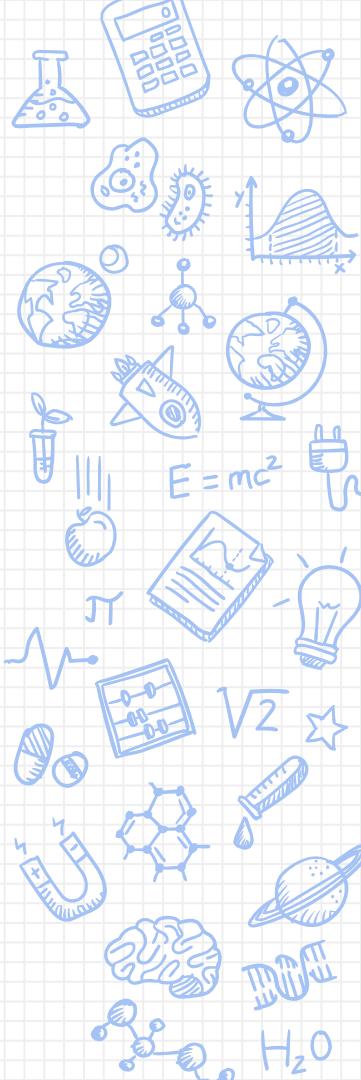


# Pip

---

> Search for a package

\$ pip search astronomy

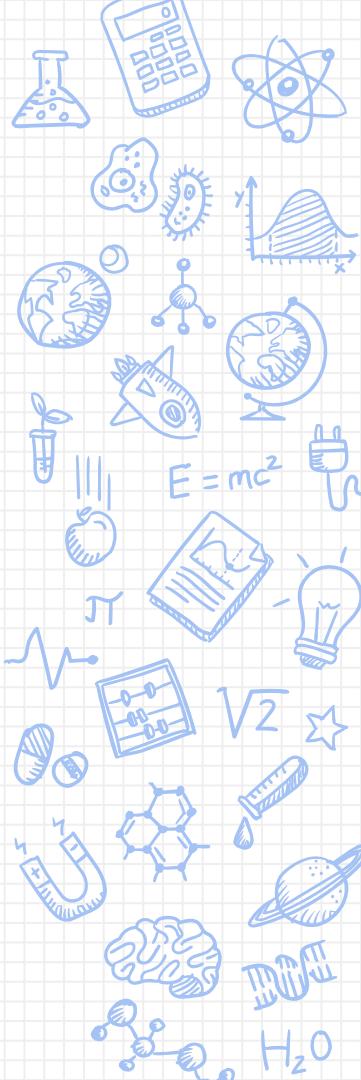


# Pip

---

> Install the latest version of a package by specifying a package's name

```
$ pip install novas
```

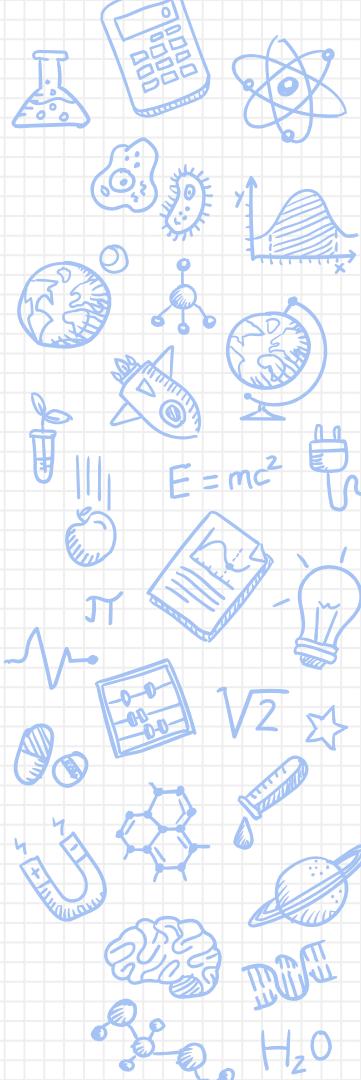


# Pip

---

> Install a specific version of a package by giving the package name followed by == and the version number

```
$ pip install requests==2.6.0
```

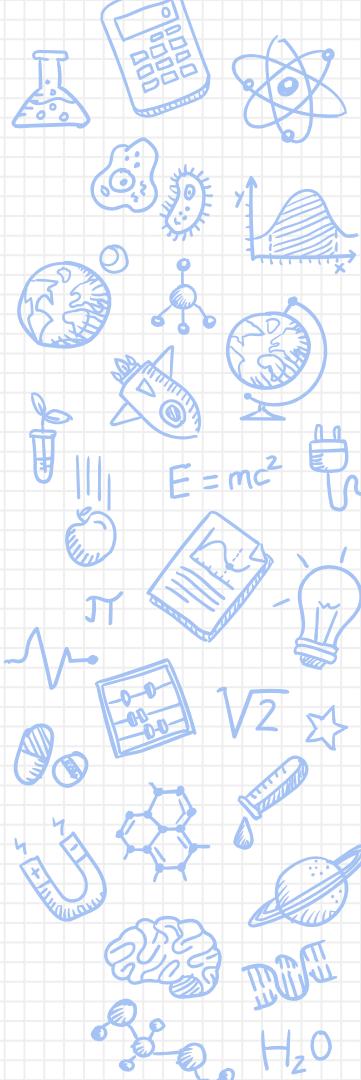


# Pip

---

> Upgrade

```
$ pip install --upgrade requests
```

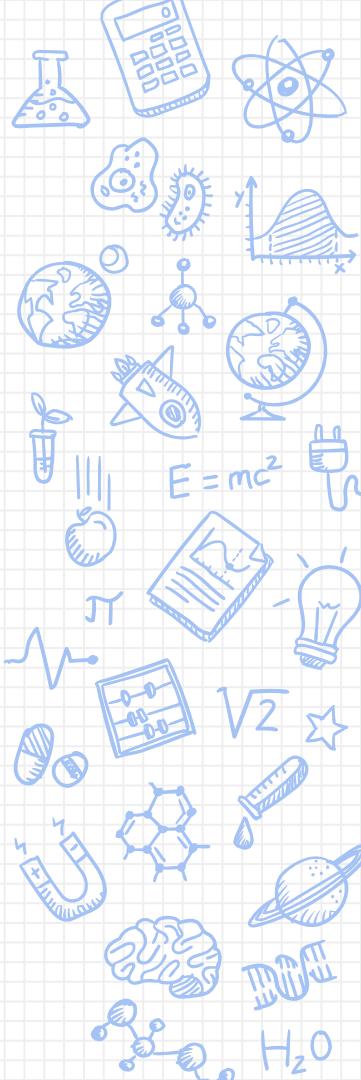


# Pip

---

> Uninstall

```
$ pip uninstall novas
```

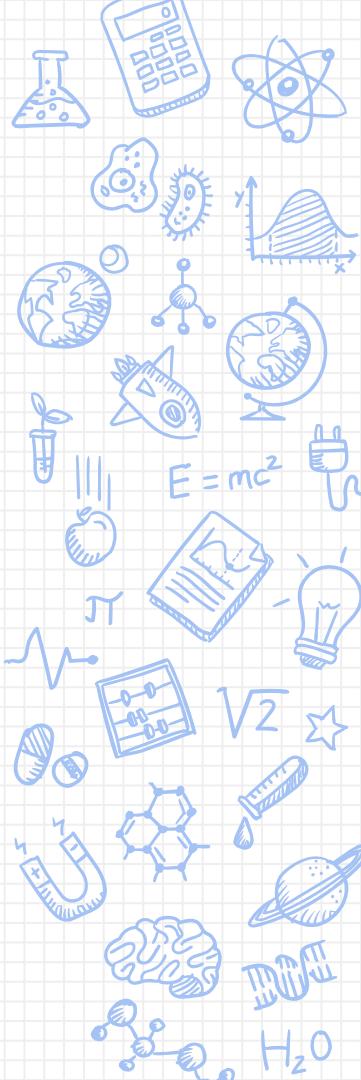


# Pip

---

> Display information about a particular package

```
$ pip show novas
```

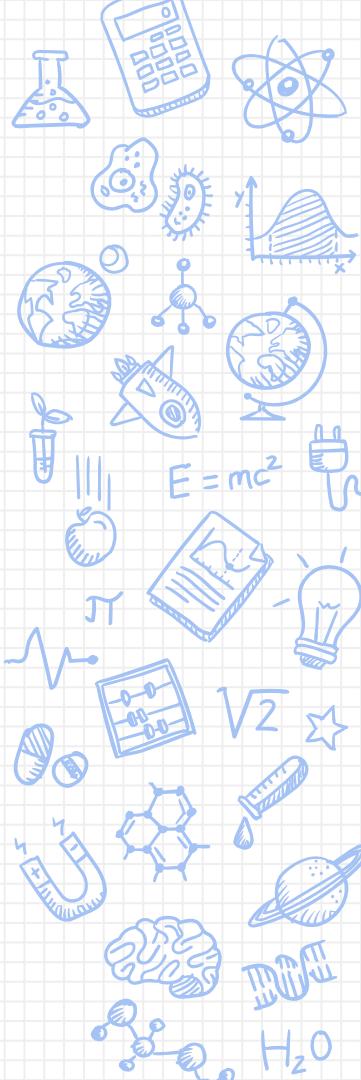


# Pip

---

> Display all of the packages installed in the environment

\$ pip list



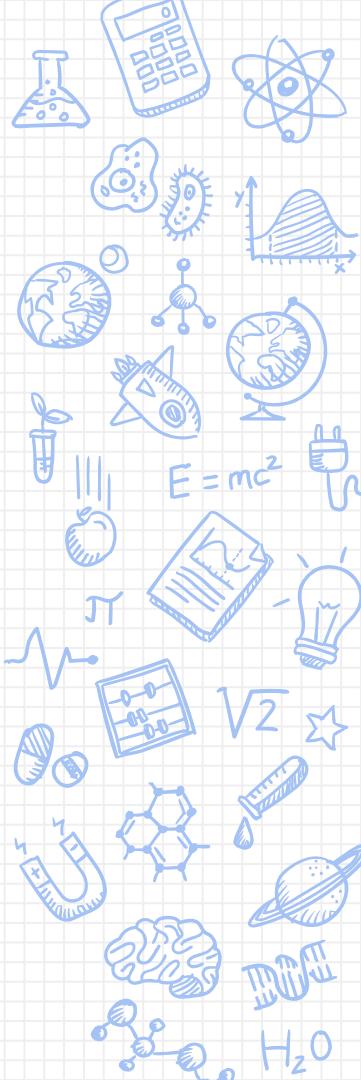
# Pip

---

> Display list of packages installed but in the 'pip install' format

```
$ pip freeze > requirements.txt
```

```
$ cat requirements.txt
```

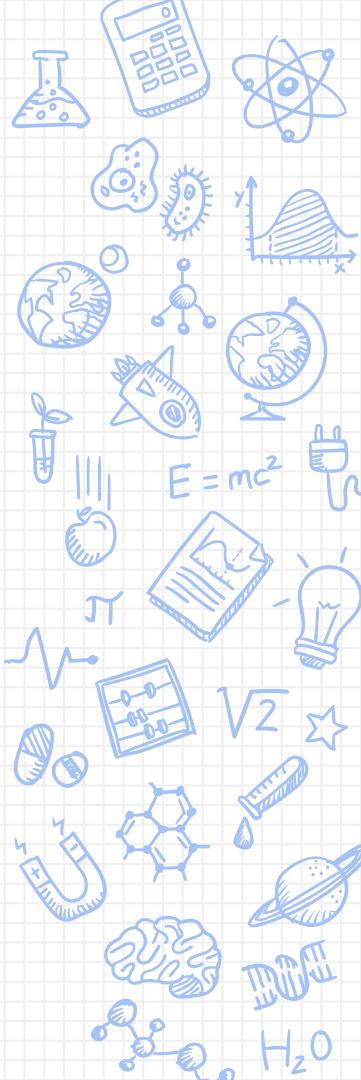


# Pip

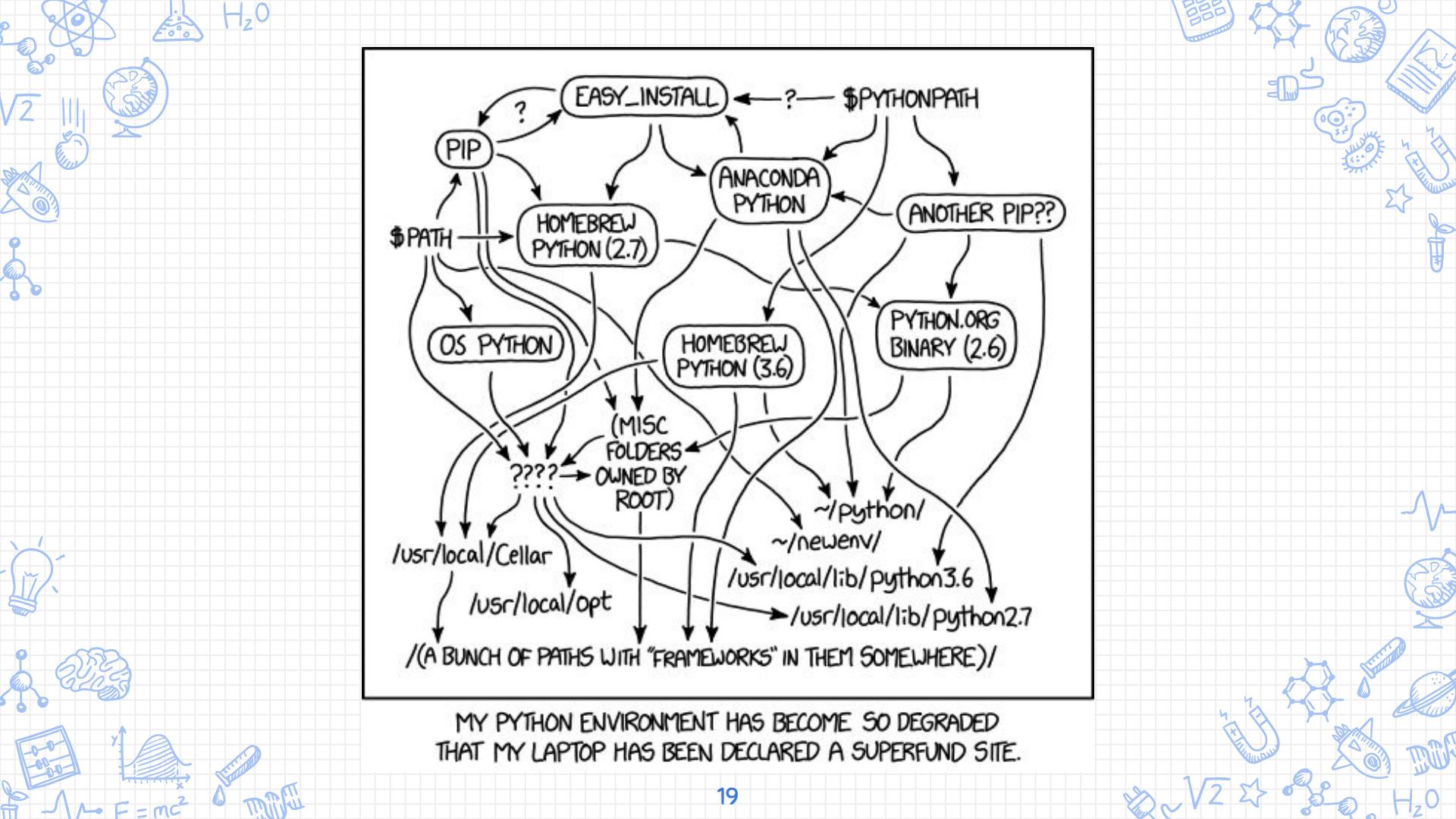
---

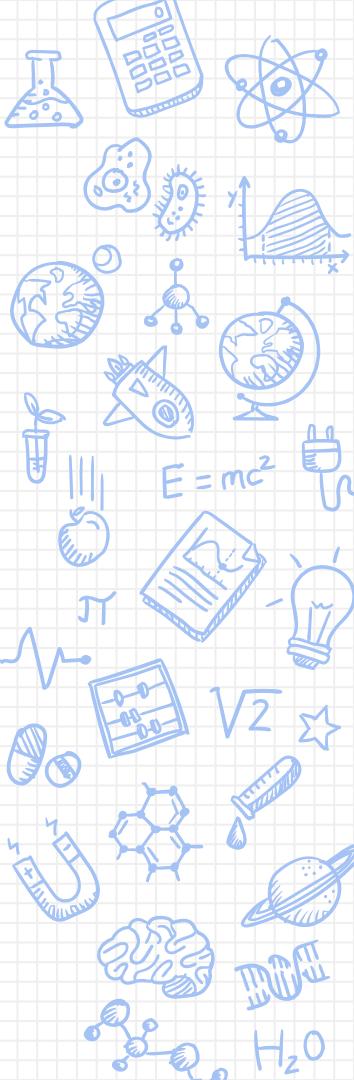
> Installation of all the necessary packages with the requirements file

```
$ pip install -r requirements.txt
```



# Why using virtual environments ?

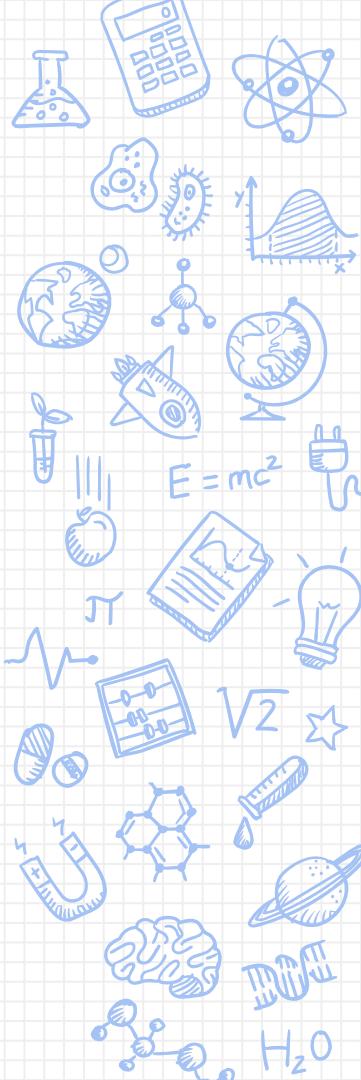




## The need for environments

---

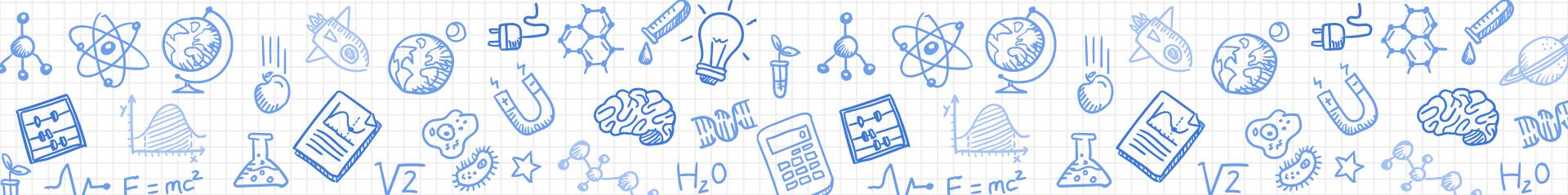
- ✗ Several versions of Python
- ✗ Specific versions of a library for a particular Python application (conflicts)
  - > With Python, all programs share the same global installation.
- ✗ Reproducibility !



# The need for environments

> Virtual environment : a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages

# Virtualenv

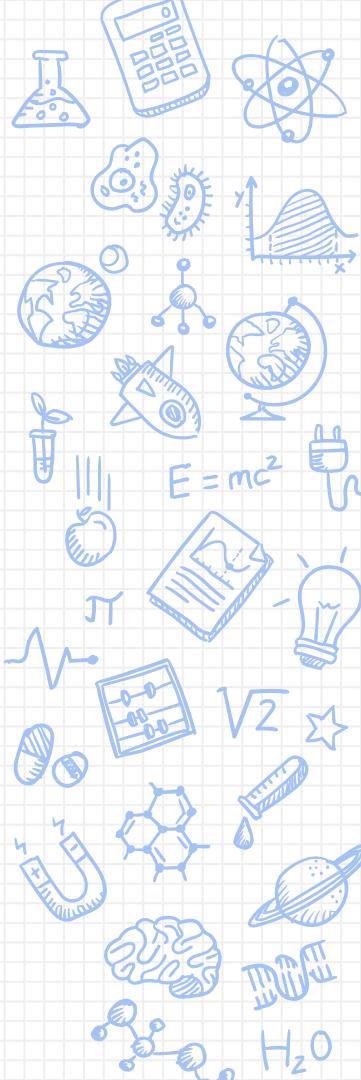


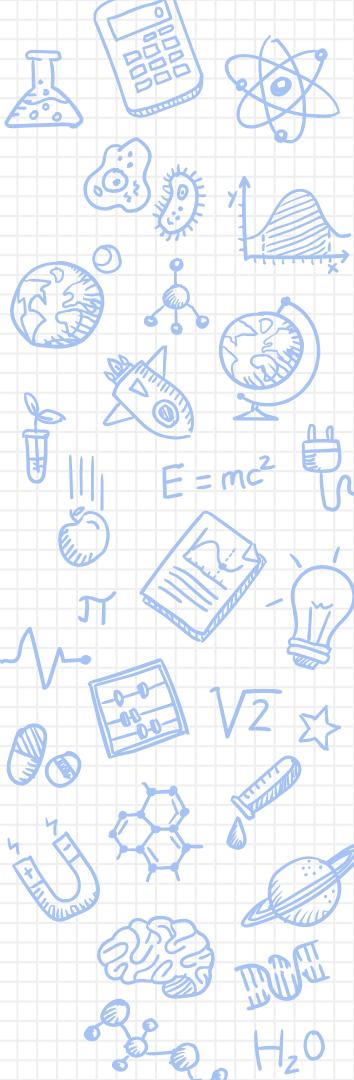
## Virtualenv

---

> a tool to create isolated Python environments.  
virtualenv creates a folder which contains all the necessary executables to use the packages that a Python project would need.

<https://pypi.org/project/virtualenv/>





## Virtualenv

---

> Create a virtual environment

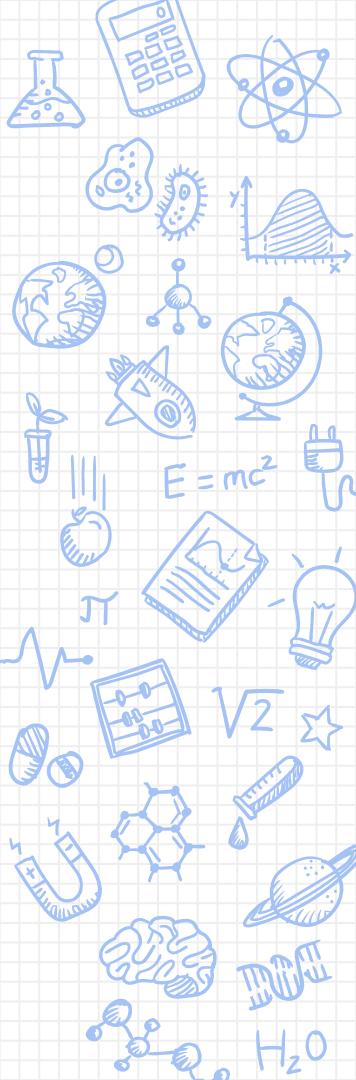
```
$ cd project_folder
```

```
$ virtualenv venv
```

# Virtualenv

> Start the virtual environment

```
$ source venv/bin/activate
```

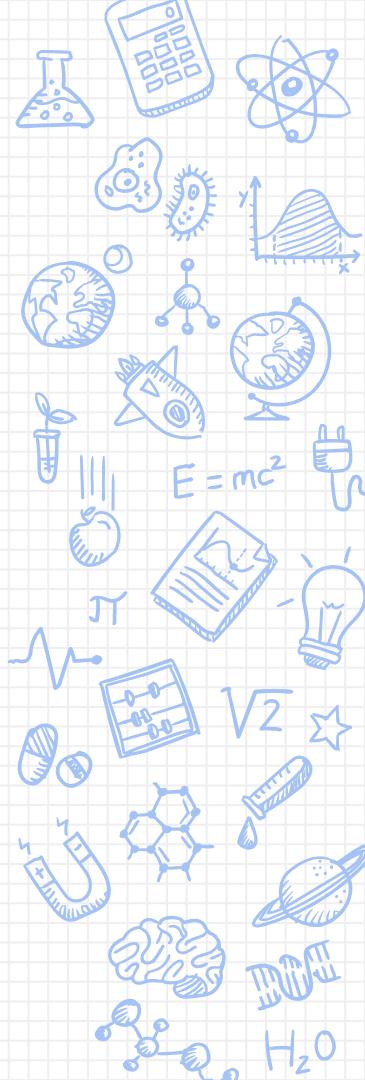


# Virtualenv

---

> Install packages with pip

\$ pip install package



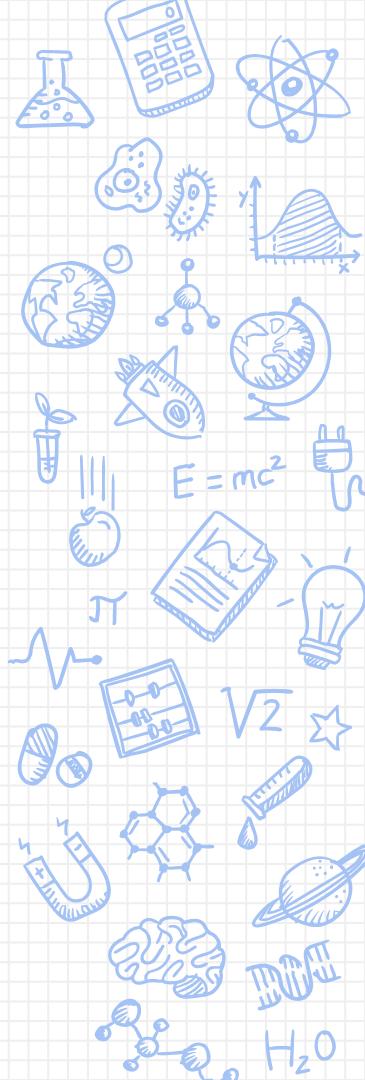
## Virtualenv

---

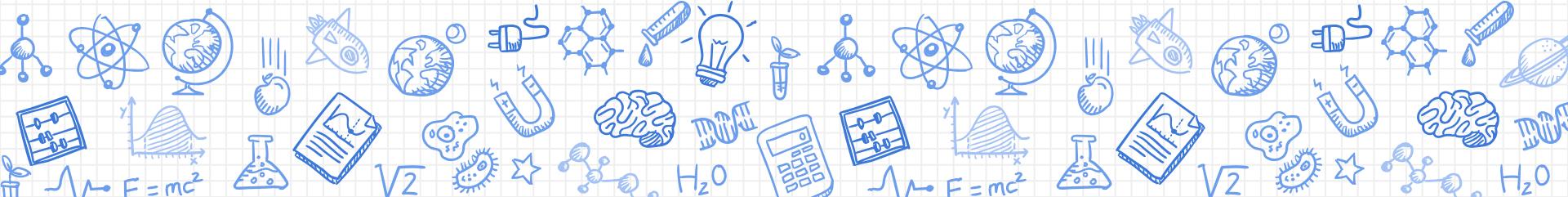
> Stop the virtual environment

\$ deactivate

> To delete the virtual environment, just delete its folder.



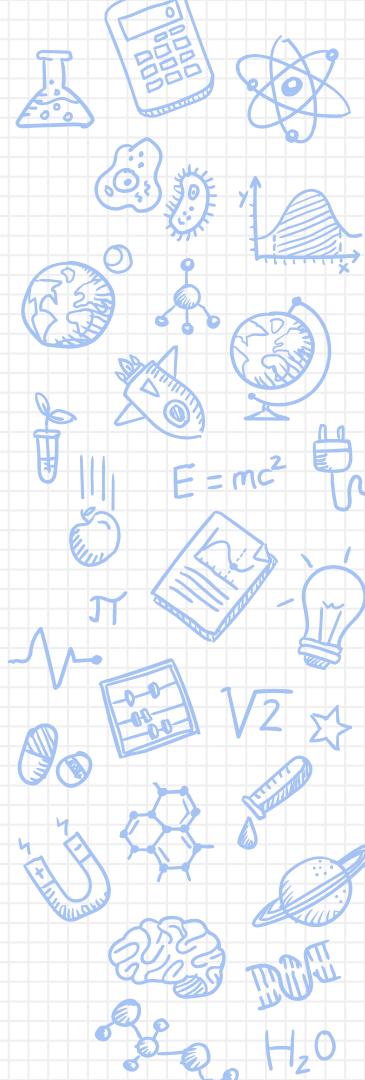
# Virtualenvwrapper



## virtualenvwrapper

---

> Provides a set of commands which makes working with virtual environments much more pleasant. It also places all your virtual environments in one place.

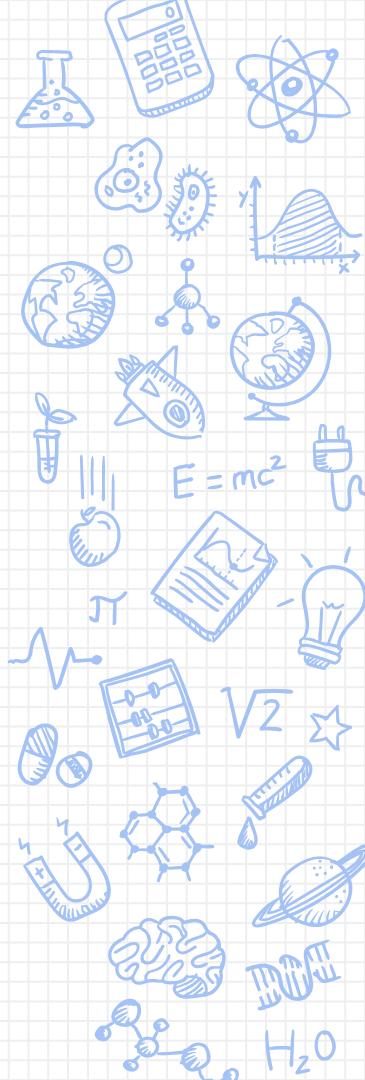


# virtualenvwrapper

---

## > Installation

```
$ pip install virtualenvwrapper  
$ export WORKON_HOME=~/Envs  
$ source /usr/local/bin/virtualenvwrapper.sh
```



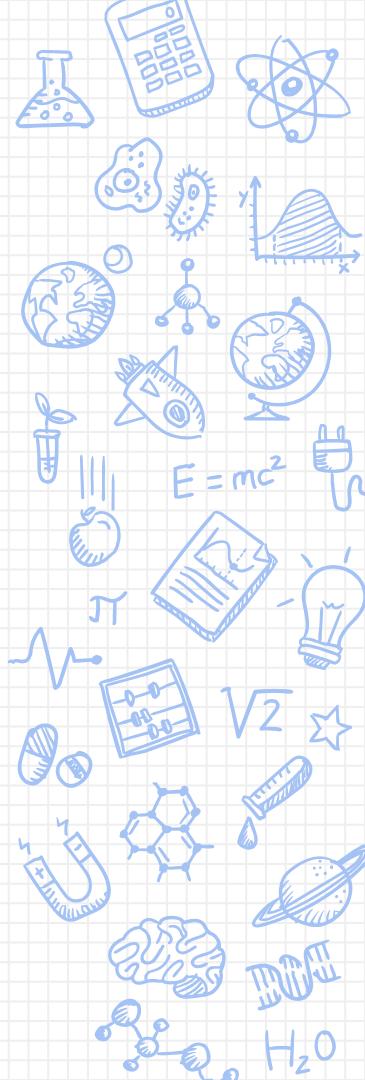
# virtualenvwrapper

---

> Create the virtual environment

```
$ mkvirtualenv project_folder
```

This creates the project\_folder folder inside  
~/Envs.

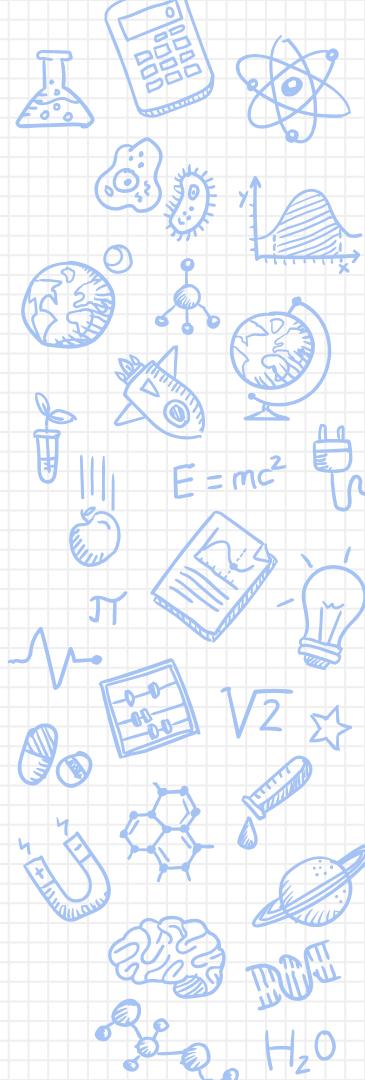


# virtualenvwrapper

---

> Work on a virtual environment

\$ workon project\_folder

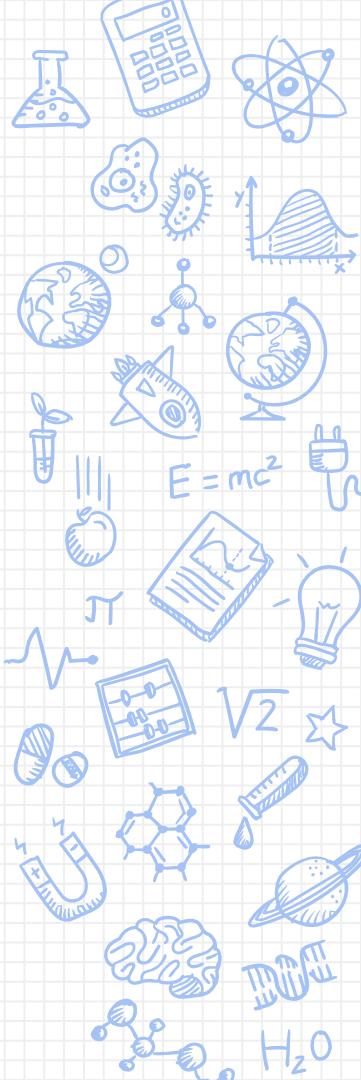


## virtualenvwrapper

---

- > Can make a project, which creates the virtual environment, and also a project directory inside \$WORKON\_HOME

```
$ mkproject project_folder
```



# virtualenvwrapper

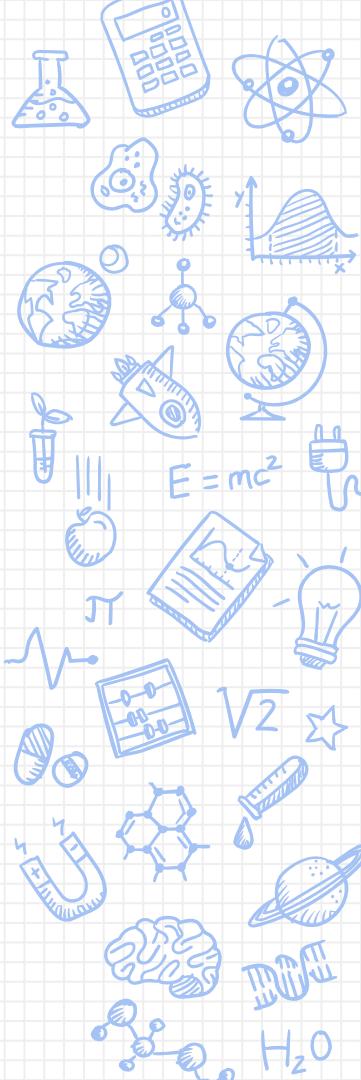
---

> Deactivate

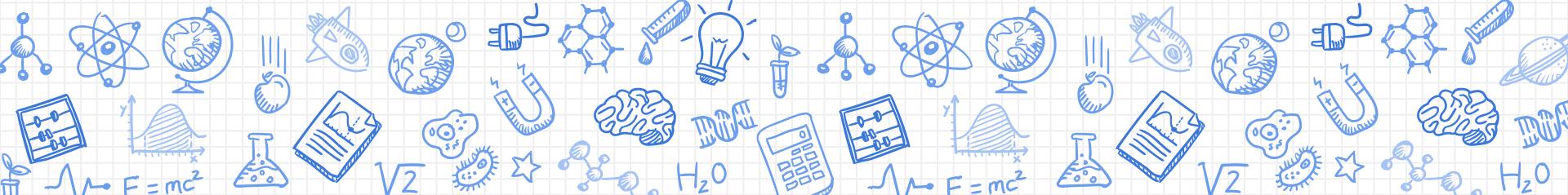
\$ deactivate

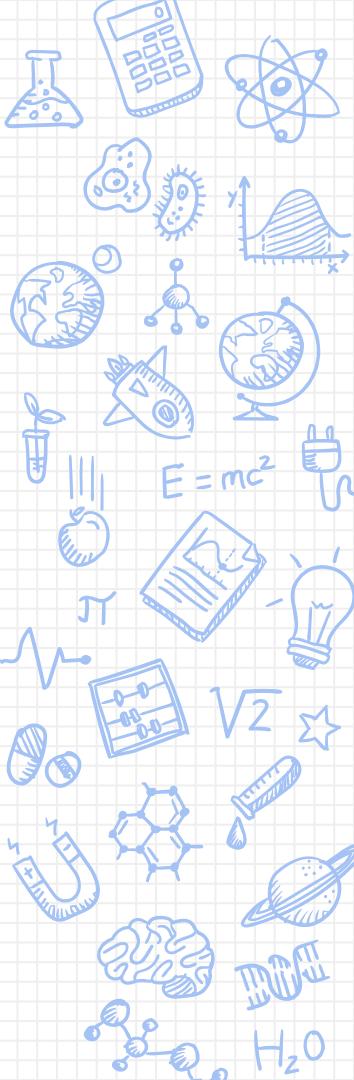
> Delete

\$ rmvirtualenv venv



# Pyenv





## Pyenv

---

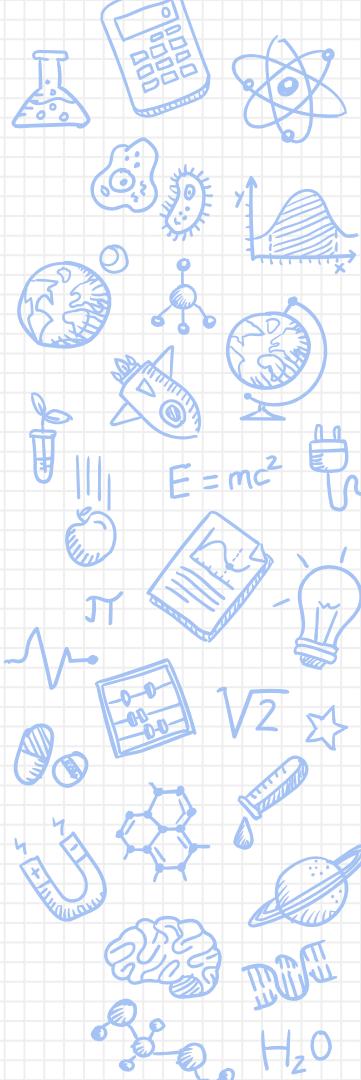
- ✗ <https://github.com/pyenv/pyenv>
- ✗ Easily switch between multiple versions of Python
- ✗ change the global Python version
- ✗ Provide support for per-project Python versions
- ✗ Allow you to override the Python version with an environment variable
- ✗ Search commands from multiple versions of Python at a time

## Pyenv

---

> Sets the global version of Python to be used in all shells

```
$ pyenv global
```

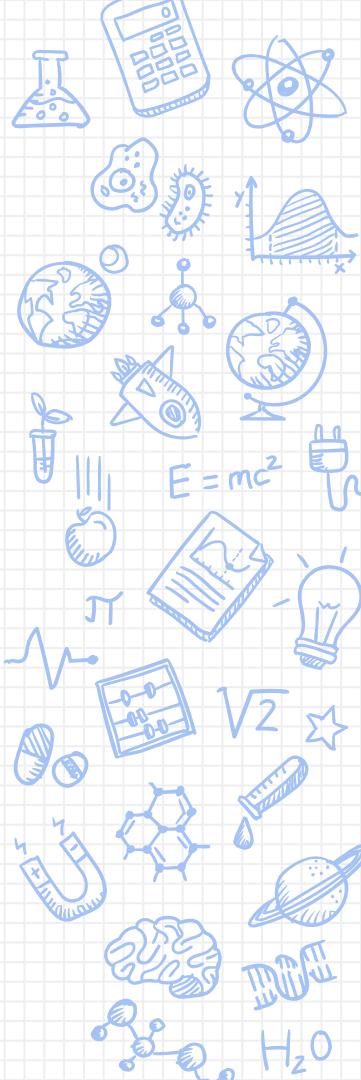


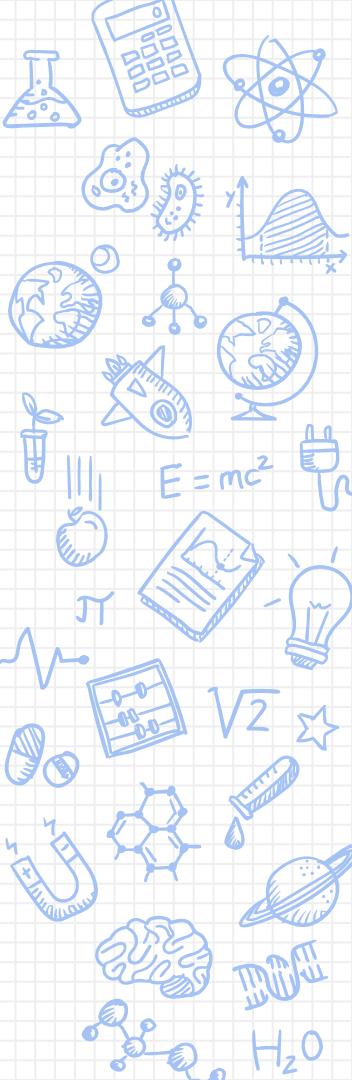
## Pyenv

---

> Sets a local application-specific Python version

```
$ pyenv local 3.7.0
```





## Pyenv

> Install a specific version

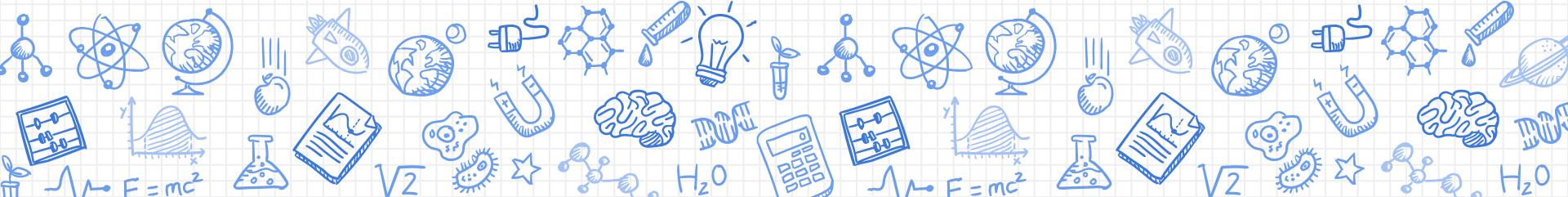
```
$ pyenv install --list
```

```
$ pyenv install --3.8.0
```

> Get installed versions

```
$ pyenv versions
```

# Pipenv

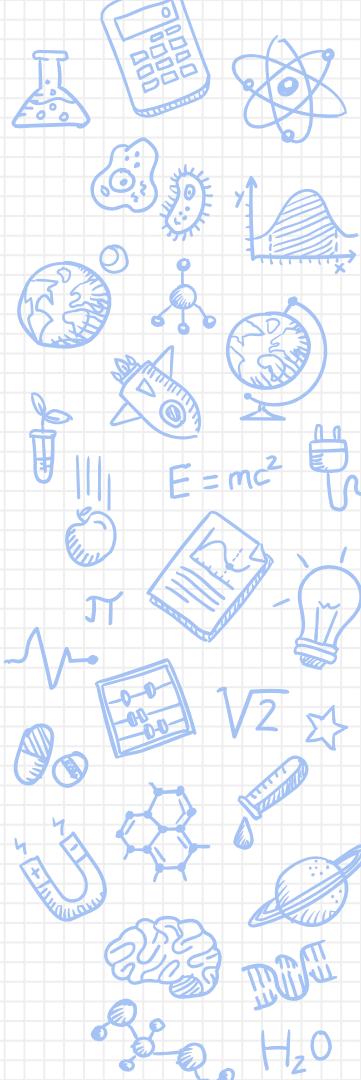


# Pipenv

---

> aims to bring the best of all packaging worlds to the Python world. It harnesses Pipfile, pip, and virtualenv into one single toolchain

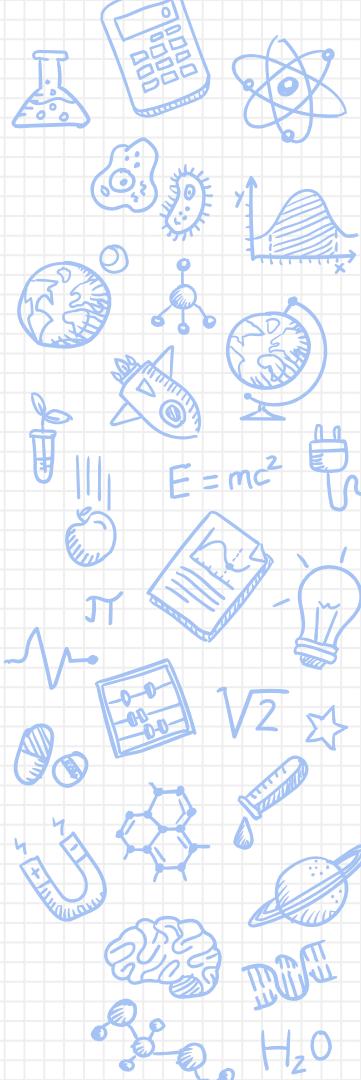
<https://pypi.org/project/pipenv/>



## Pipenv

---

- > don't need to manipulate your pipfile > pipenv does it for you
- > combination of virtualenv + pip
- > recommended by PyPA (Python Packaging Authority)
- > use it to generate and both create env, install dependencies into env, update pipfile / lock file.



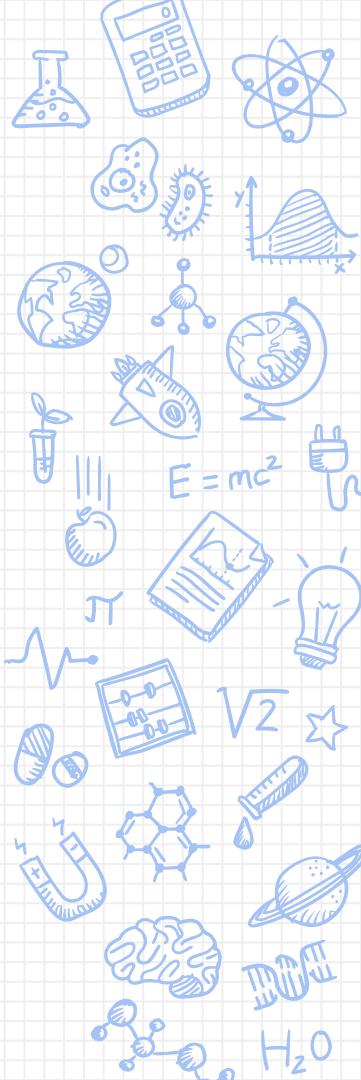
## Pipenv

---

> relies on two separate files :

- pipfile
  - similar to requirements.in -> requirements.txt
  - human readable
- pipfile.lock
  - real requirements.txt

> pipfile + lock pipfile : used to regenerate env

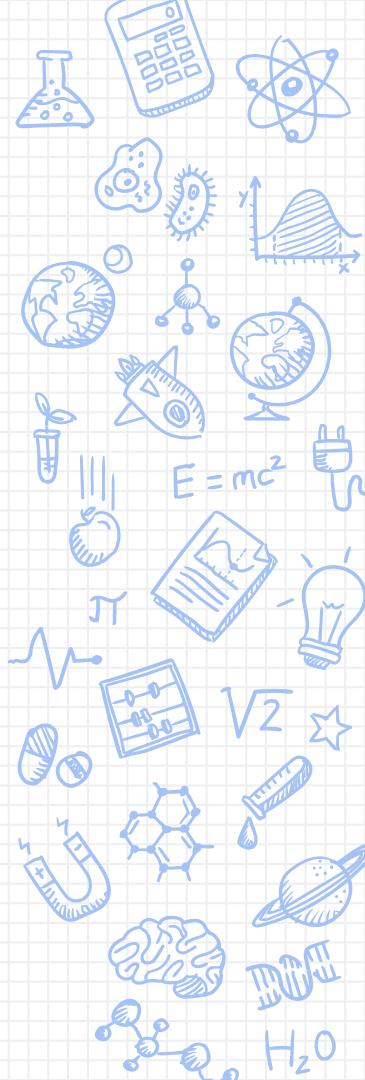


## Pipenv

---

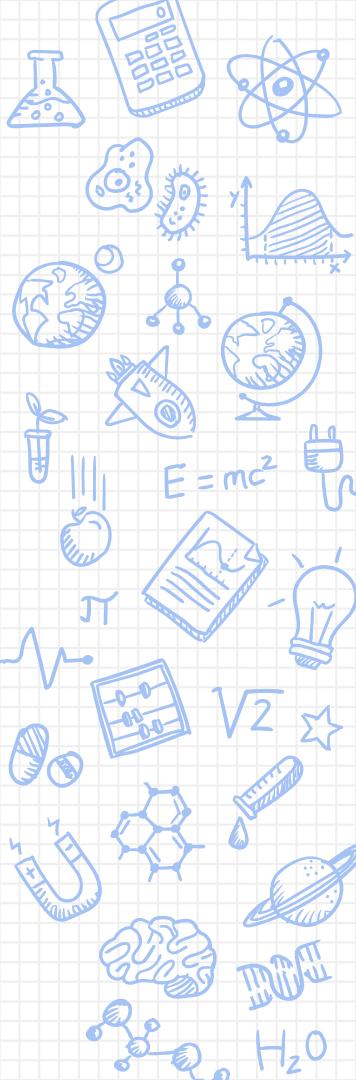
`pipenv install package` > install package + update  
pipfile & lock file

`pipenv install --dev package` >> separate packages dev  
/ production / etc



# Pipenv

- > pipenv shell : activating the virtual env, spawns a sub-shell inside your shell, run in the virtual env
  - > pipenv run python script.py : run the command using the virtual env but only the command. Back to original prompt /shell
  - > pipenv graph : generate graph of packages installed and their dependencies, shows it

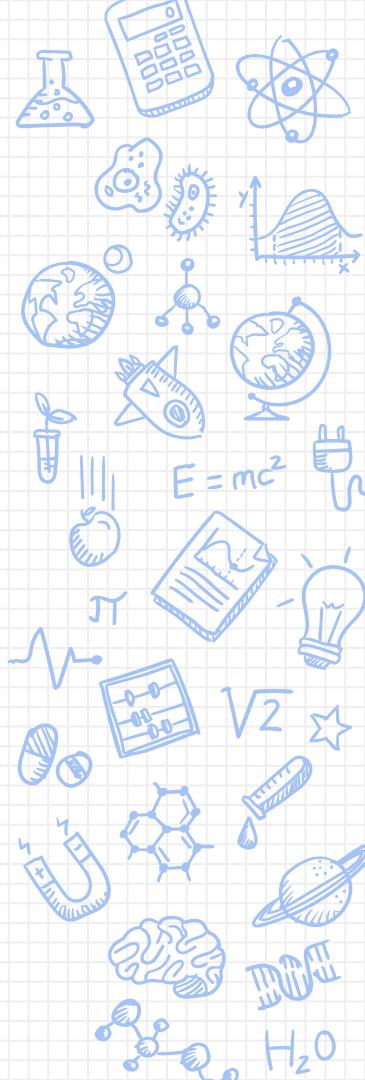


## Pipenv

---

Drawbacks :

- not recommended if you are creating a package/library. Only for programs
- locking can be slow
- by default, update all the dependencies every time a new package is installed
  - can be changed with --selective-upgrade

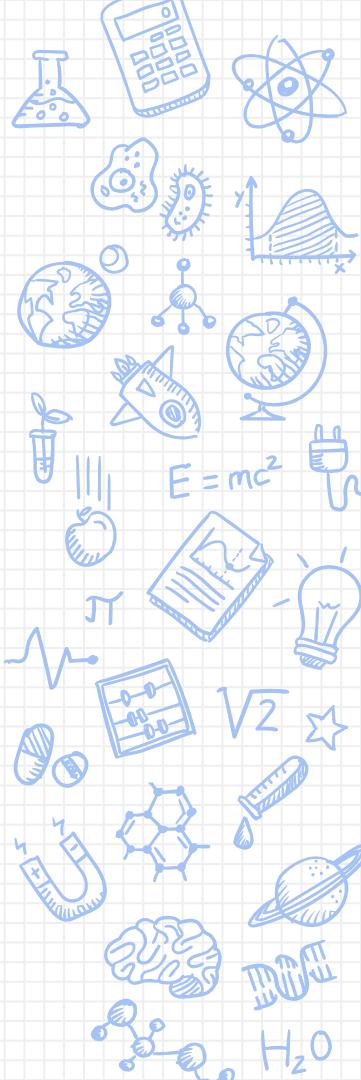


# Pipenv

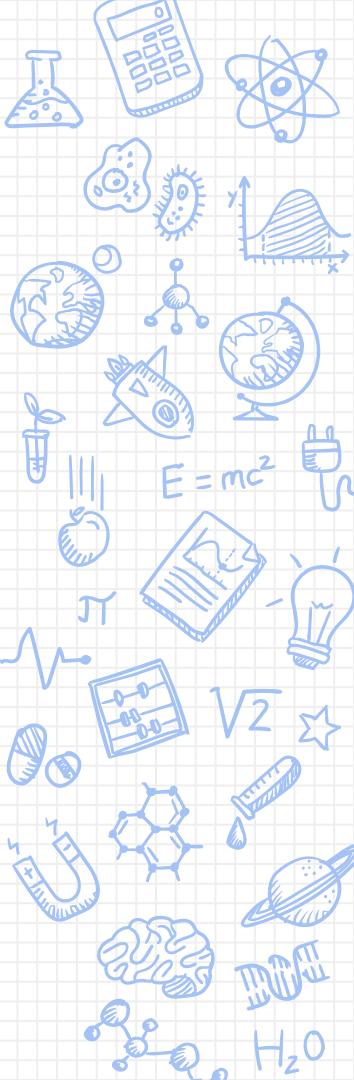
---

Pros:

- installs packages and updates lock file in one command
- installs packages concurrently
- prevents dependency conflict
- check for security



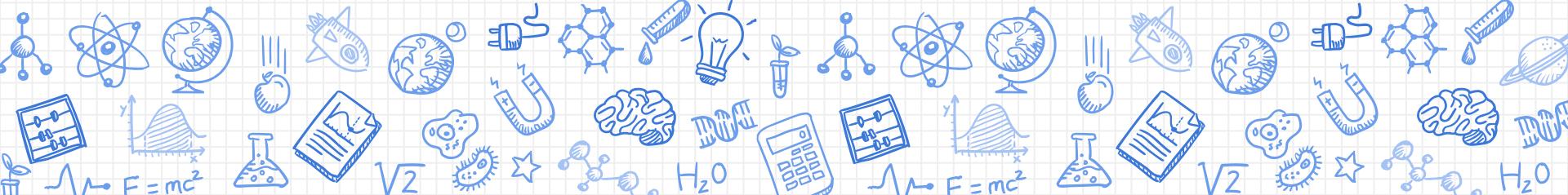
# Invoking Python

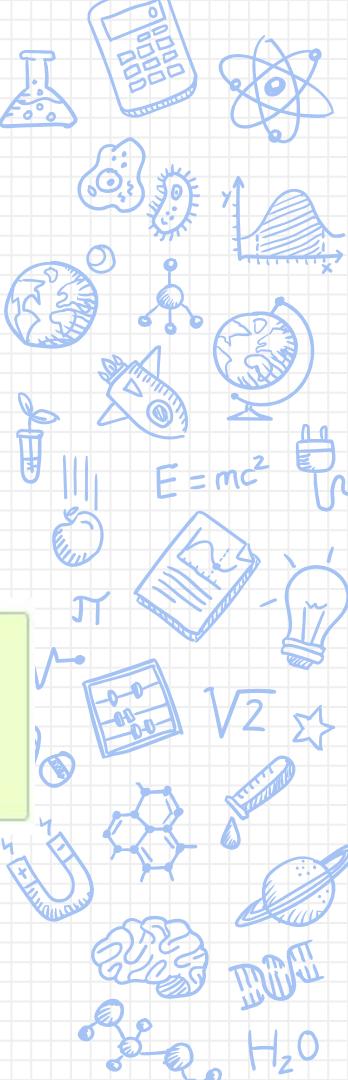


## Invoking Python

- ✗ By running a python file
  - ✗ e.g. `python myscript.py`
- ✗ Through an interactive console
  - ✗ Python Interpreter or IPython shell
- ✗ In an interactive notebook

# The Python Interpreter





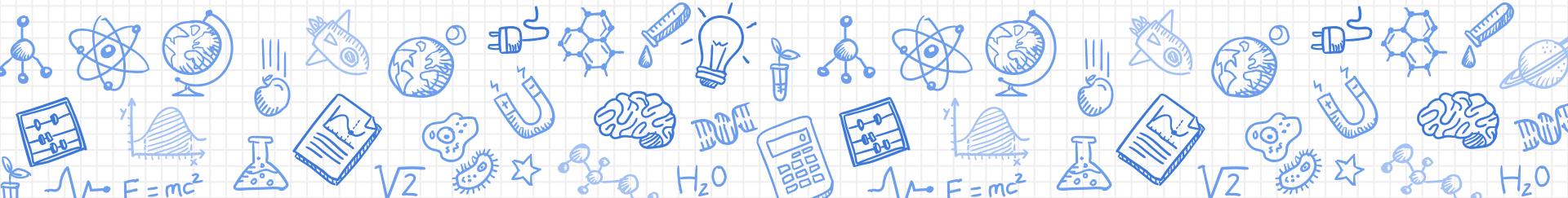
# The Python Interpreter

- ✗ Usually located in '/usr/local/bin' (Linux)
- ✗ 'Python' in shell
- ✗ Interactive mode

```
$ python3.8
Python 3.8 (default, Sep 16 2015, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

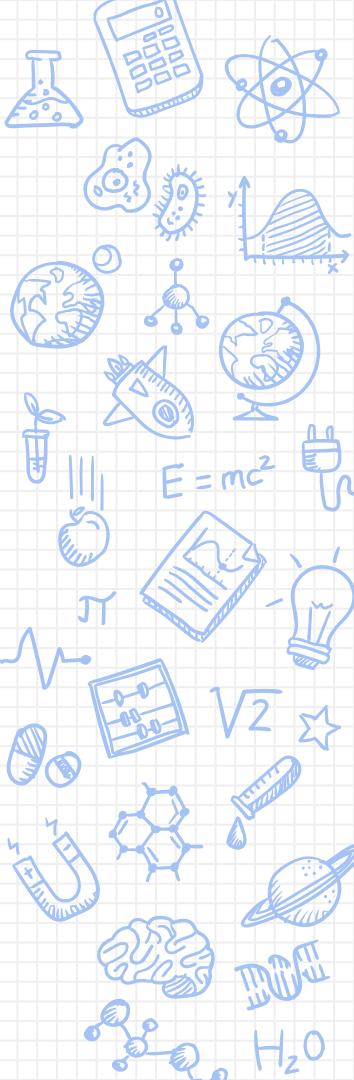
- ✗ Exit with 'quit()'

# IPython



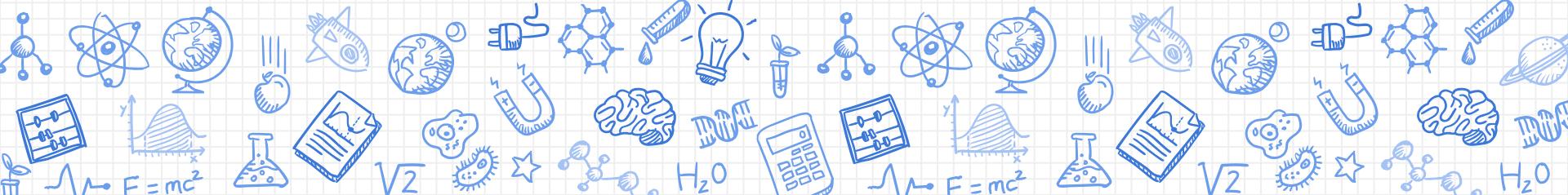
# IPython

---



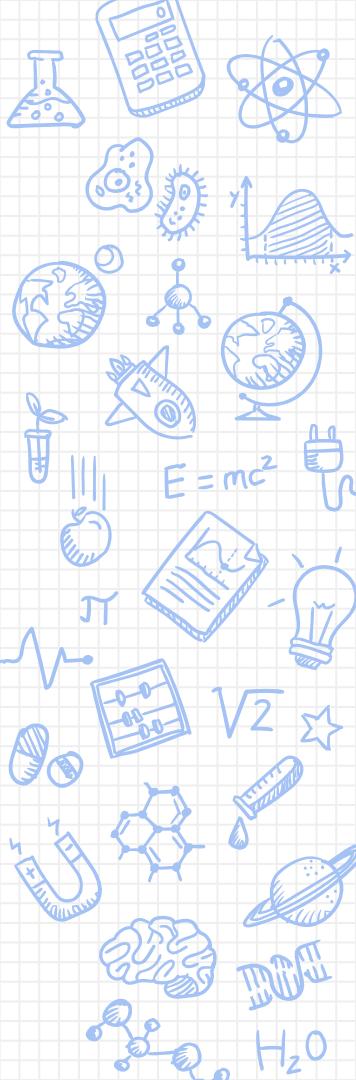
- ✗ developed by Fernando Perez in 2001 as an enhanced Python interpreter
- ✗ Syntax highlighting
- ✗ Tab completion of keywords, variables and function names
- ✗ Stores the history of interactions
- ✗ Provides access to Python debugger
- ✗ Acts as a main kernel for Jupyter notebook

# Anaconda / Conda



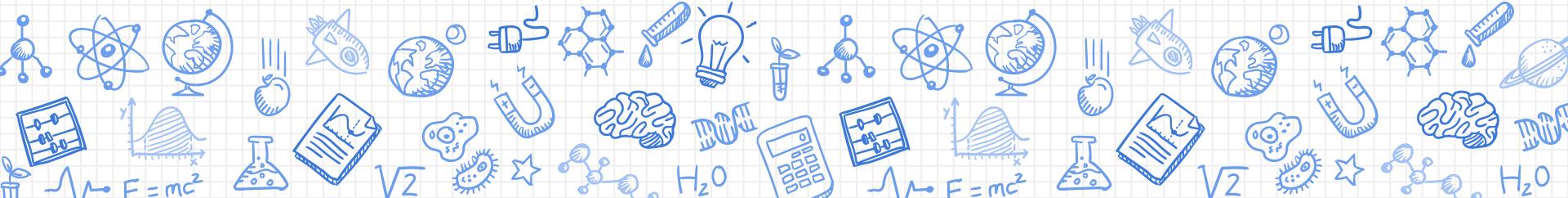
# Anaconda

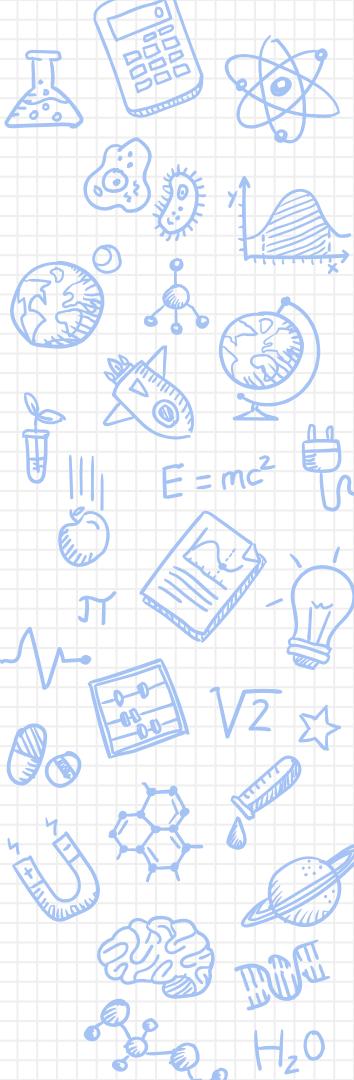
- popular in the scientific community
  - combination of package manager (like pip) & env manager (like virtualenv + virtualenvwrapper)
  - easier to use ? (considered)
  - does not support all the packages that pip does (e.g beautifulsoup)



# Jupyter Notebook

Demo + TP

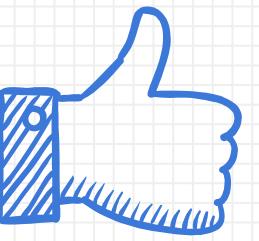




## References

---

- ✗ [https://nbviewer.jupyter.org/github/rabernat/python\\_teaching/blob/master/one\\_day\\_workshop/01\\_core\\_python.ipynb](https://nbviewer.jupyter.org/github/rabernat/python_teaching/blob/master/one_day_workshop/01_core_python.ipynb)
- ✗ <https://docs.python.org/3.8/tutorial/>
- ✗ [https://www.youtube.com/watch?v=i7\\_US0Jff30](https://www.youtube.com/watch?v=i7_US0Jff30)
- ✗ [https://www.tutorialspoint.com/jupyter/ipython\\_getting\\_started.htm](https://www.tutorialspoint.com/jupyter/ipython_getting_started.htm)
- ✗ <https://docs.python-guide.org/dev/virtualenvs/>



# THANKS!

Any questions?