

# Programación básica en Python



**Aarón Velasco Agustin**

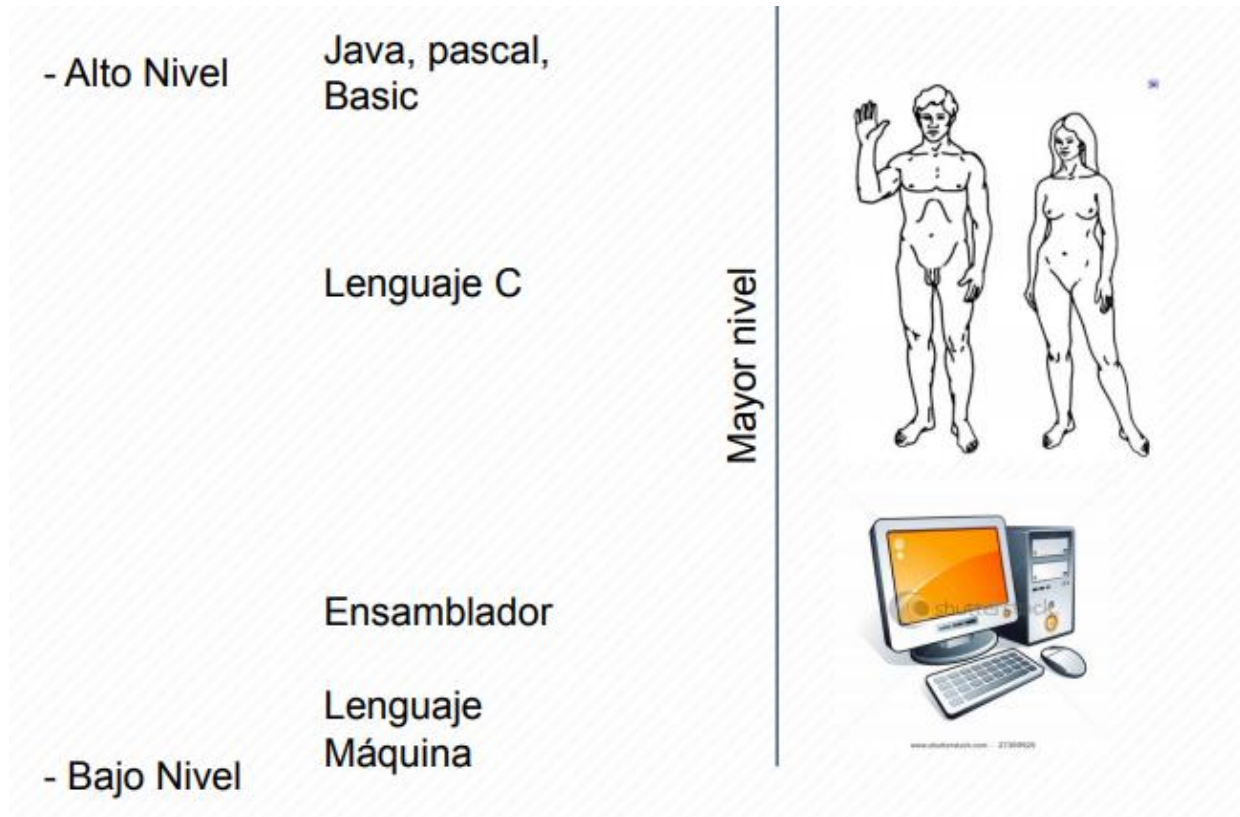
# ¿Que es un lenguaje de programación?

- Es un lenguaje artificial
- Posee reglas sintácticas y semánticas
- Sirve para escribir instrucciones para una computadora, para lo cual requiere ser traducido, es decir, compilado o interpretado.



```
254
255 function updatePhotoDescription() {
256     if (descriptions.length > (page * 9) + (currentImage subtring() - 1)) {
257         document.getElementById( bigimageDesc ).innerHTML = descriptions[page * 9 + (currentImage subtring() - 1)];
258     }
259 }
260
261 function updateAllImages() {
262     var i = 1;
263     while (i < 10) {
264         var elementId = 'foto' + i;
265         var elementIdBig = 'bigimage' + i;
266         if (page * 9 + i - 1 < photos.length) {
267             document.getElementById( elementId ).src = 'images/' + photos[page * 9 + i - 1];
268             document.getElementById( elementIdBig ).src = 'images/' + photos[page * 9 + i - 1];
269         } else {
270             document.getElementById( elementId ).src = '';
```

# Lenguajes de bajo, medio y alto nivel.



- ▶ **Lenguaje de bajo nivel:** es el que proporciona poca o ninguna abstracción del microprocesador de un ordenador. Consecuentemente es fácilmente trasladado a lenguaje de máquina.
  - ▶ En general se utiliza este tipo de lenguaje para programar controladores (drivers).
- ▶ **Lenguaje de medio nivel** es un lenguaje de programación informática como el lenguaje C, que se encuentran entre los lenguajes de alto nivel y los lenguajes de bajo nivel.
  - ▶ Suelen ser clasificados muchas veces de alto nivel, pero permiten ciertos manejos de bajo nivel. Son precisos para ciertas aplicaciones como la creación de sistemas operativos, ya que permiten un manejo abstracto (independiente de la máquina, a diferencia del ensamblador), pero sin perder mucho del poder y eficiencia que tienen los lenguajes de bajo nivel.

- Los lenguajes de alto nivel se caracterizan por expresar los algoritmos de una manera adecuada a la capacidad cognitiva humana, en lugar de a la capacidad ejecutora de las máquinas.



A word cloud featuring various programming languages and frameworks. The words are arranged in a circular pattern, tilted at different angles. The colors of the text vary, including red, purple, blue, green, and yellow. The words include: Perl, C#, Python, RUBY ON RAILS, C, JavaScript, PHP, AJAX, Java, and VB. Net.



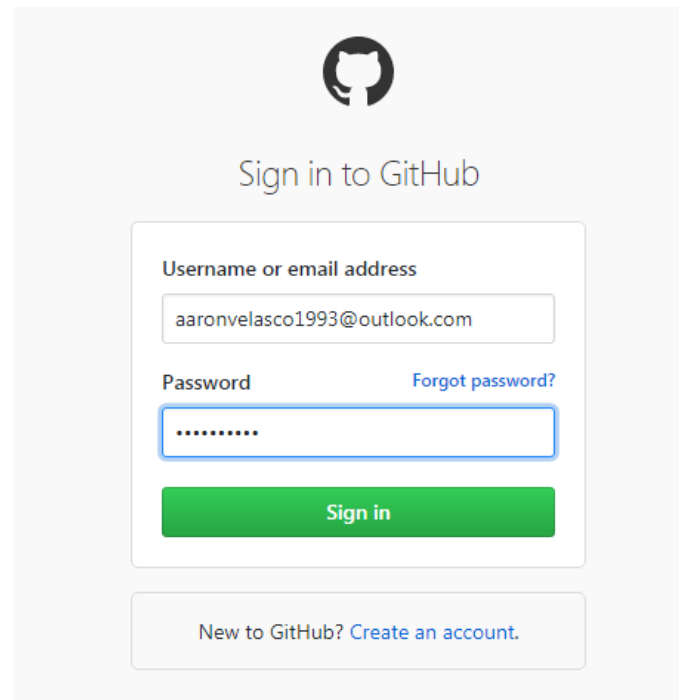
# GitHub

# Repositorio (GitHub)

- ▶ Que es: Es un archivo donde se depositan (nube), en formato digital, materiales derivados de la producción científica o académica de una institución (universidades, centros de investigación).
- ▶ **Objetivo:** Facilitar el acceso de la comunidad científica a los resultados de la investigación realizada por sus miembros y aumentar la visibilidad de la producción científica de la institución.
- ▶ **Git Hub**
- ▶ GitHub aloja tu repositorio de código y te brinda herramientas muy útiles para el trabajo en equipo, dentro de un proyecto.
- ▶ Además de eso, puedes **contribuir a mejorar el software de los demás**
- ▶ En nuestra especialidad "*Programación*", fuimos aprendiendo cosas y creando programas de código abierto, fomentando el **software libre**; es por eso que presentamos esta gran herramienta enfocada al **crecimiento de proyectos comunitarios y libres**.

# Git Hub

- ▶ Ingresar a la siguiente liga: <https://github.com/>
- ▶ Registro en la pagina.
- ▶ Inicar sesión.



The image shows the GitHub login interface. At the top is the GitHub logo (Octocat). Below it is the text "Sign in to GitHub". The form contains two input fields: "Username or email address" with the value "aaronvelasco1993@outlook.com" and "Password" with masked characters "\*\*\*\*\*". There is a link "Forgot password?" next to the password field. Below the fields is a green "Sign in" button. At the bottom, there is a link "New to GitHub? Create an account."

Sign in to GitHub

Username or email address  
aaronvelasco1993@outlook.com

Password [Forgot password?](#)  
\*\*\*\*\*

Sign in

New to GitHub? [Create an account.](#)





Search GitHub

[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



Browse activity

Discover repositories



msveronikee started following you

16 days ago



Adrian0910 started following you

17 days ago

Repositories



OmarRm2

0 ★



OmarRm2

0 ★



Darthsoviet/computadoras-y-pro...

0 ★



antonio221/curshtml

0 ★

New repository

Import repository

New gist

New organization

# Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



AaronVelasco93 ▾

/

Repository name

PythonCurso



Great repository names are short and memorable. Need inspiration? How about **cautious-guacamole**.

Description (optional)

Repositorio para curso de Python



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.



Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

Repositorio para curso de Python

Edit

[Add topics](#)

1 commit

1 branch

0 releases

1 contributor

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾

AaronVelasco93 Initial commit Latest commit ef5534b just now

README.md Initial commit just now

README.md

# PythonCurso

---

Repositorio para curso de Python

---

<> Code

! Issues 0

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

📊 Insights

⚙️ Settings

PythonCurso /



Drag files here to add them to your repository

Or [choose your files](#)



## Commit changes

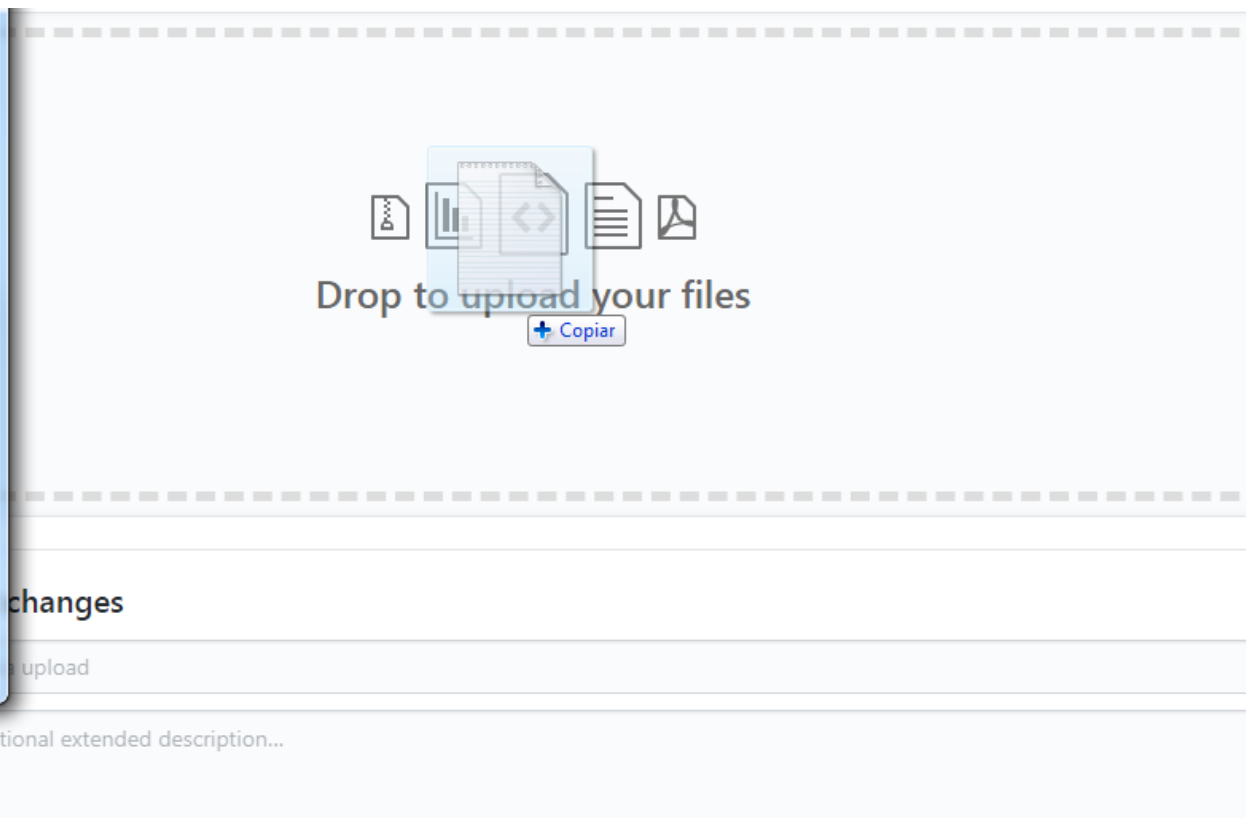
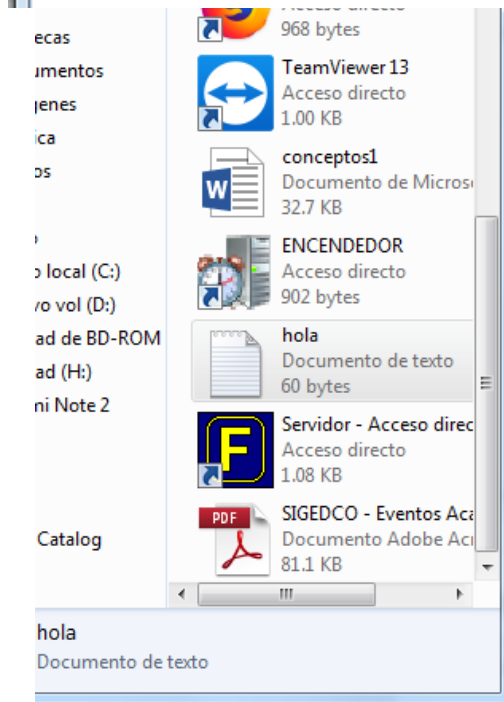
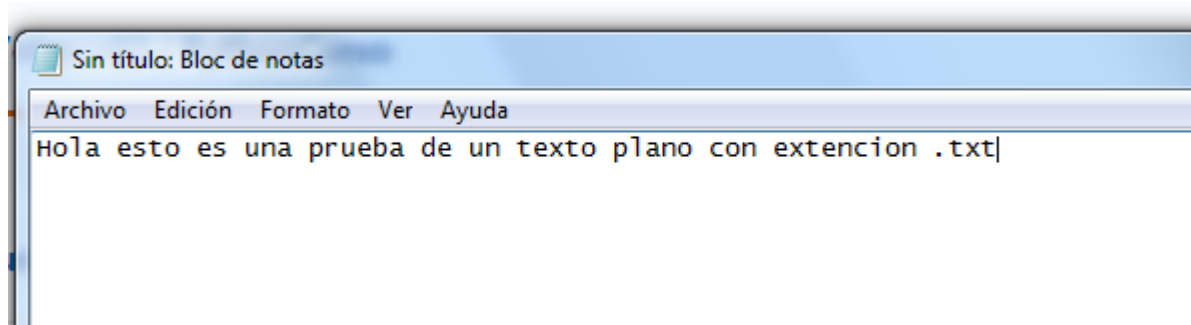
Add files via upload

Add an optional extended description...

- ☒ Commit directly to the `master` branch.
- ☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes

Cancel



hola.txt



## Commit changes

documento

se guarda un documento .txt prueba de repositorio

- ☒ Commit directly to the `master` branch.
- ☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes

Cancel

Branch: master ▾

New pull request



AaronVelasco93 documento ...

README.md

hola.txt

README.md

## PythonCurso

Repositorio para curso de Python

Branch: master ▾

PythonCurso / hola.txt

Find file

Copy path



AaronVelasco93 documento

23a2fa3 43 seconds ago

1 contributor

1 lines (1 sloc) | 60 Bytes

Raw

Blame

History



1 Hola esto es una prueba de un texto plano con extension .txt

1 ■■■■■ hola.txt

... @@ -1 +0,0 @@

1 -Hola esto es una prueba de un texto plano con extension .txt



### Commit changes

.

.

☒ Commit directly to the `master` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more](#)

Commit changes

Cancel

## Danger Zone

### Make this repository private

Please [upgrade your plan](#) to make this repository private.

### Transfer ownership

Transfer this repository to another user or to an organization where you have the ability to create repositories.

[Transfer](#)

### Archive this repository

Mark this repository as archived and read-only.

[Archive this repository](#)

### Delete this repository

Once you delete a repository, there is no going back. Please be certain.

[Delete this repository](#)

### Are you absolutely sure?



Unexpected bad things will happen if you don't read this!

This action cannot be undone. This will permanently delete the `AaronVelasco93/PythonCurso` repository, wiki, issues, and comments, and remove all collaborator associations.

Please type in the name of the repository to confirm.

[I understand the consequences, delete this repository](#)[Create new file](#)[Upload files](#)[Find file](#)[Clone or download ▾](#)

### Clone with HTTPS ?

[Use SSH](#)

Use Git or checkout with SVN using the web URL.

[Open in Desktop](#)[Download ZIP](#)

4 months ago



# Términos de Programación

```
def get_connections(self, user):  
    """  
    Returns a QuerySet of connections for user.  
    """  
    set1 = self.filter(from_user=user).select_related(depth=1)  
    set2 = self.filter(to_user=user).select_related(depth=1)  
    return set1 | set2  
  
def are_connected(self, user1, user2):  
    if self.filter(from_user=user1, to_user=user2).count() > 0:  
        return True  
    if self.filter(from_user=user2, to_user=user1).count() > 0:  
        return True  
    return False  
  
def remove(self, user1, user2):  
    """  
    Deletes proper object regardless of the order of users in argument  
    """  
    connection = self.filter(from_user=user1, to_user=user2)  
    if not connection:  
        connection = self.filter(from_user=user2, to_user=user1)  
    connection.delete()  
    -----  
models.py      Top L1      (Python AC yas)-----
```

# Sintaxis

- ▶ Se trata de la rama de la gramática que ofrece pautas creadas para saber cómo unir y relacionar palabras a fin de elaborar oraciones y expresar conceptos de modo coherente.
- ▶ En la informática, la sintaxis se entiende como el grupo de normas que marcan las secuencias correctas de los elementos propios de un lenguaje de programación.

- Declaración:
  - a = valor
- Condiciones:
  - if a == b and c == d:  
    print 'Hola'
  - elif a != d:  
    print 'Nada que hacer'
  - else:  
    print 'Chao'

# Compilador e interprete

Un compilador es aquel programa que:

- ▶ □ Traduce un programa escrito en un lenguaje de alto nivel a otro de bajo nivel, que usualmente resulta ser lenguaje máquina.
- ▶ □ La aplicación funciona una vez que no existen errores de sintaxis, los errores lógicos se verán en su ejecución.
- ▶ □ Una vez compilado, se genera un archivo ejecutable (.exe), que funcionará cada vez que se requiera.

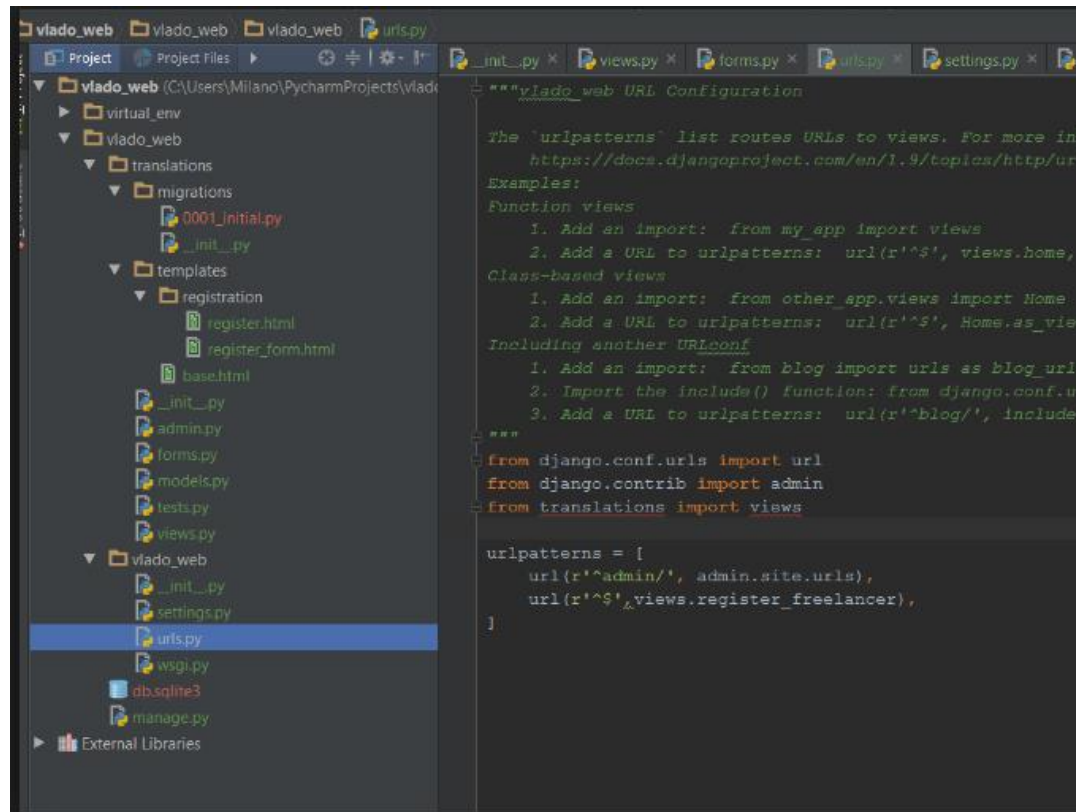
IDE-> para Python ->Pycharm IDE

Un intérprete es un programa que:

- ▶ □ Analiza y ejecuta simultáneamente código de un lenguaje fuente, así como revisar variables de entornos e interrumpirse en cualquier momento.
- ▶ □ Corre línea por línea, por lo que encontrar errores es algo complicado, ya que se presentarán al llegar a la línea que no se ejecute correctamente.
- ▶ (Python)

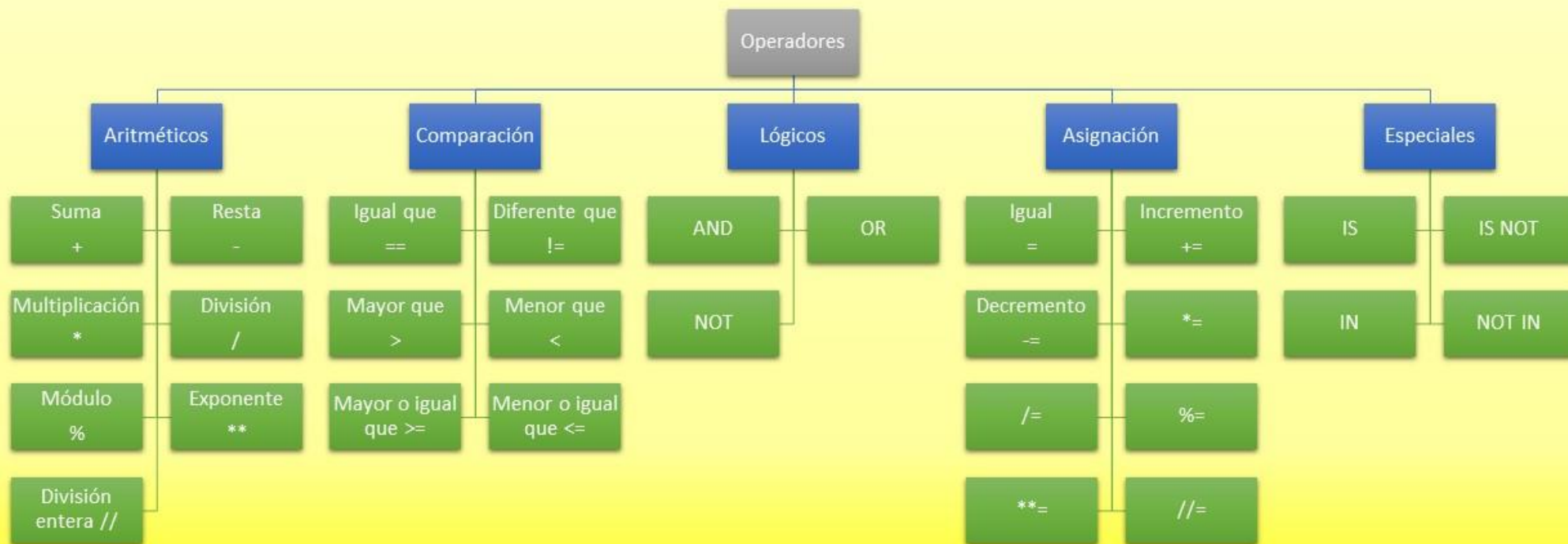
# IDE

- Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.



# Tipos y operadores





Símbolo	Significado	Ejemplo	Resultado
+	Suma	<code>a = 10 + 5</code>	<code>a es 15</code>
-	Resta	<code>a = 12 - 7</code>	<code>a es 5</code>
-	Negación	<code>a = -5</code>	<code>a es -5</code>
*	Multiplicación	<code>a = 7 * 5</code>	<code>a es 35</code>
**	Exponente	<code>a = 2 ** 3</code>	<code>a es 8</code>
/	División	<code>a = 12.5 / 2</code>	<code>a es 6.25</code>
//	División entera	<code>a = 12.5 / 2</code>	<code>a es 6.0</code>
%	Módulo	<code>a = 27 % 4</code>	<code>a es 3</code>

# Que es Python

- ▶ Python es un lenguaje de programación interpretado, una de sus ventajas es el manejo de un código legible y sencillo. Esto es, la lógica de programación puede ser utilizada para aplicarse con otros lenguajes realizando cambios mínimos en los mismos.
- ▶ A finales de los años 80's, Python fue creado por Guido van Rossum en el Centro para las Matemáticas y la Informática en los Países Bajos. Su nombre proviene de la afición del creador por los humoristas británicos Monty Python.



# Características

Tipado dinámico: La mayoría de lenguajes debe pasar por dos etapas para declarar variables:

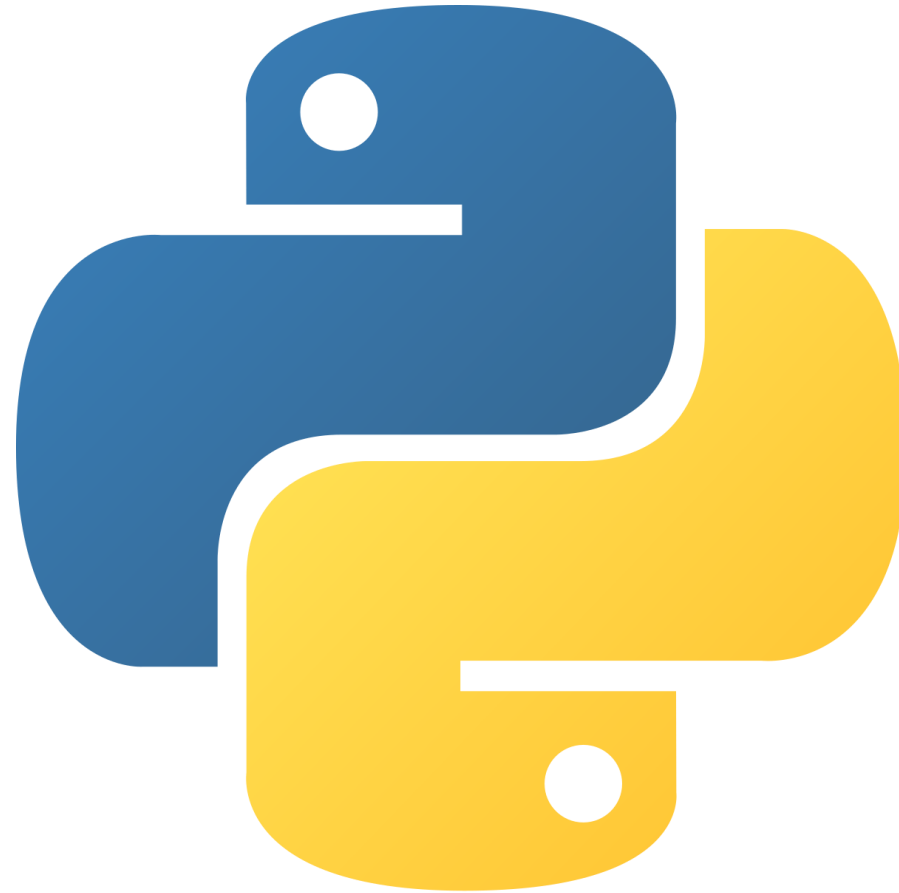
1. Colocar el nombre y el tipo de dato de la variable.
  2. Asignar un valor de acuerdo al tipo de dato especificado anteriormente.
- Sin embargo, Python realiza estas dos etapas en una sola, esto es, el tipo de dato se establece de forma automática dependiendo del dato asignado. Además, es capaz de modificar el tipo posteriormente si se requiere

Indentación: Para delimitar bloques de código, como sentencias, funciones o bucles, Python utiliza dos puntos “:” como delimitadores al inicio de su cuerpo

Multiparadigma: Permite que los programadores adopten diferentes estilos de programación:

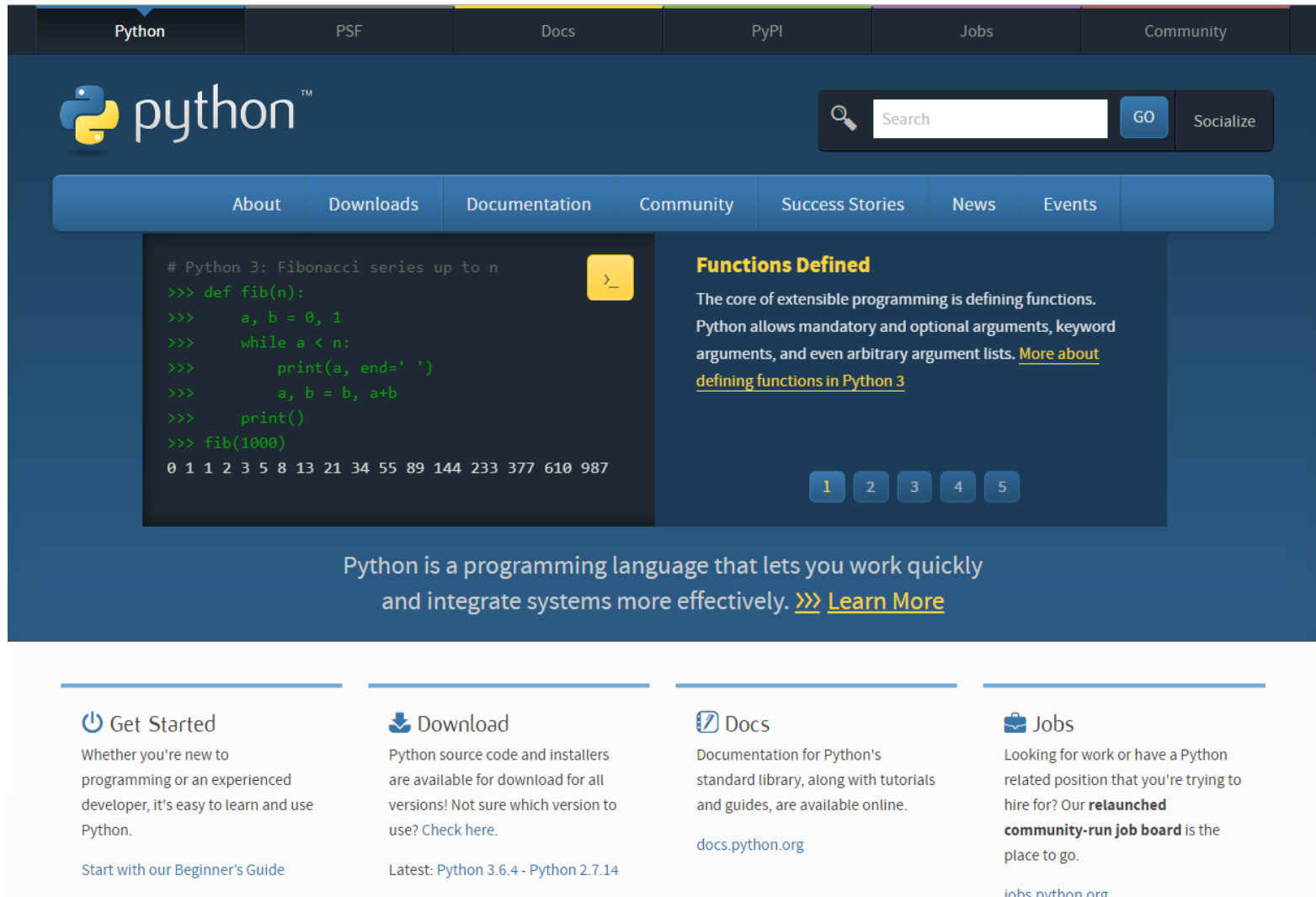
- ▶ □ Programación Estructurada
- ▶ □ Programación Funcional
- ▶ □ Programación Orientada a Objetos

# Id de Python y que versión



# Id de Python y que versión

<https://www.python.org/>



The screenshot shows the Python.org homepage with a dark blue header and a light blue main content area. The header includes navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is a search bar and a 'Socialize' button. The main content area features a navigation bar with links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The central content area is divided into two columns. The left column displays a Python 3 code snippet for a Fibonacci series, with a yellow button containing a prompt character '>\_'. The right column is titled 'Functions Defined' and contains text about the core of extensible programming, with a link to 'More about defining functions in Python 3'. Below this text are five numbered buttons (1-5). At the bottom of the main content area, a blue banner states: 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'. The footer section contains four columns: 'Get Started' with a power icon and a link to the Beginner's Guide; 'Download' with a download icon and a link to check the latest versions; 'Docs' with a book icon and a link to the documentation; and 'Jobs' with a briefcase icon and a link to the community-run job board.

Python

PSF

Docs

PyPI

Jobs

Community

python™

Search

GO

Socialize

About

Downloads

Documentation

Community

Success Stories

News

Events

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

**Functions Defined**

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. [More about defining functions in Python 3](#)

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

**Get Started**

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

[Start with our Beginner's Guide](#)

**Download**

Python source code and installers are available for download for all versions! Not sure which version to use? [Check here](#).

Latest: [Python 3.6.4](#) - [Python 2.7.14](#)

**Docs**

Documentation for Python's standard library, along with tutorials and guides, are available online.

[docs.python.org](https://docs.python.org)

**Jobs**

Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go.

[jobs.python.org](https://jobs.python.org)

Downloads

Documentation

Community

Success Stories

News

All releases

Source code

Windows

Mac OS X

Other Platforms

License

Alternative Implementations

## Download for Windows

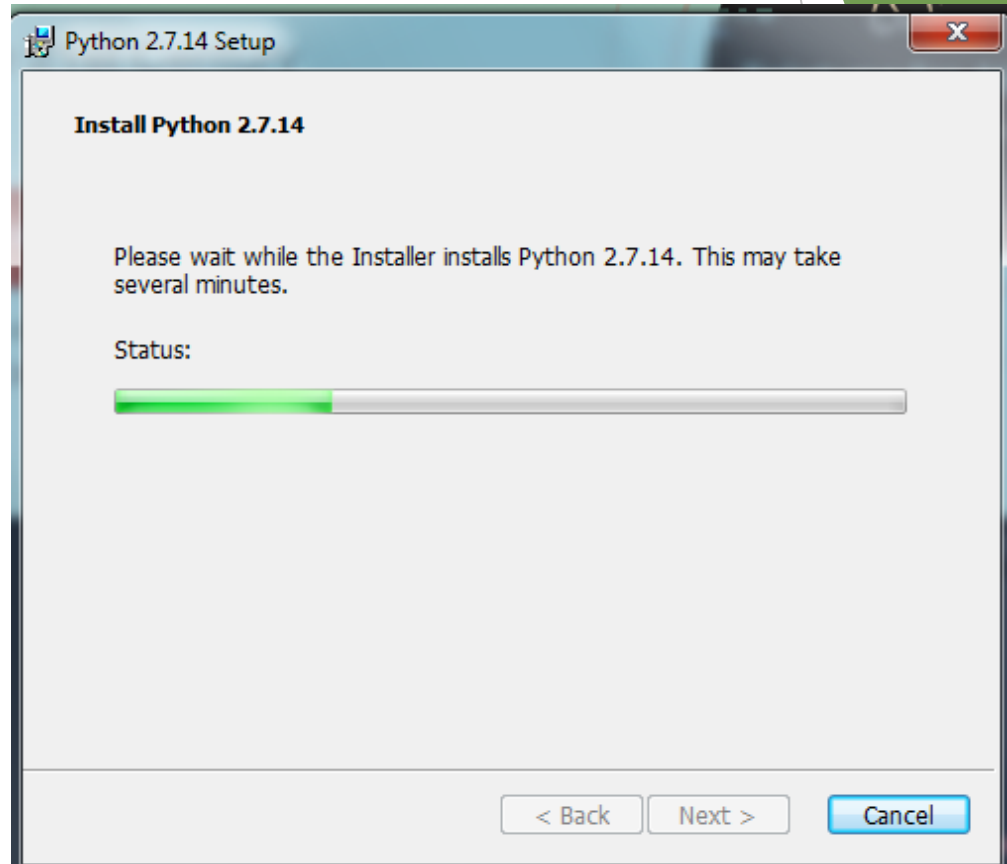
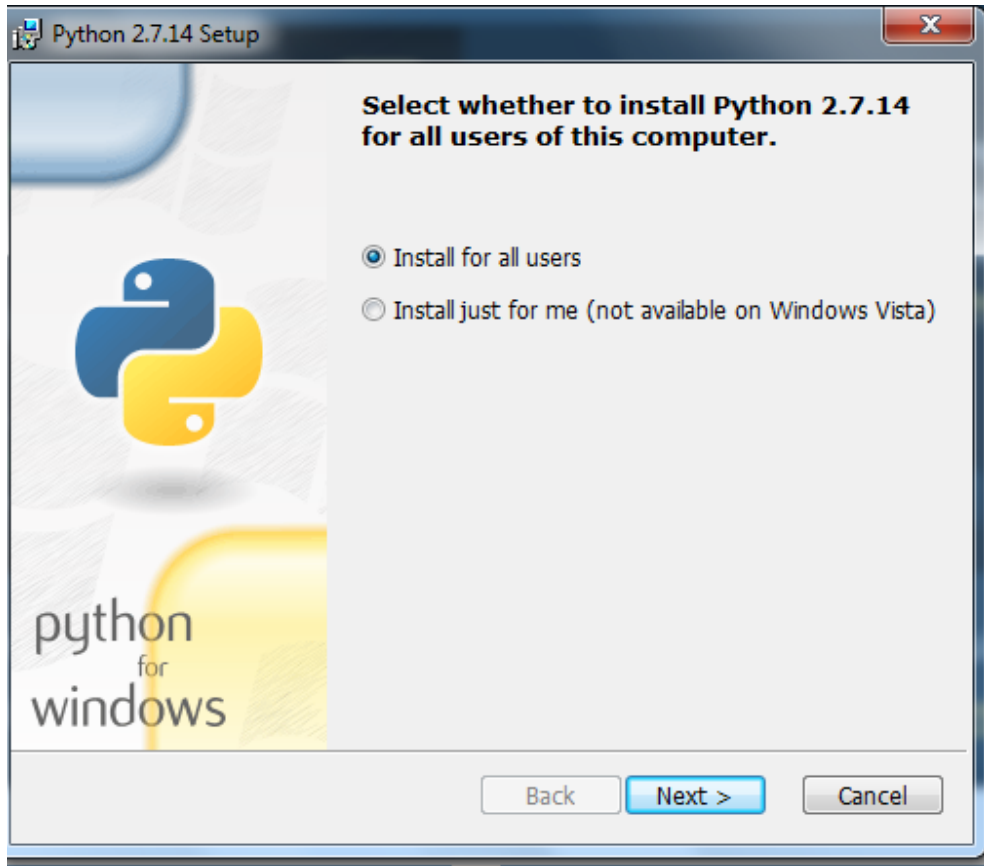
Python 3.6.4

Python 2.7.14

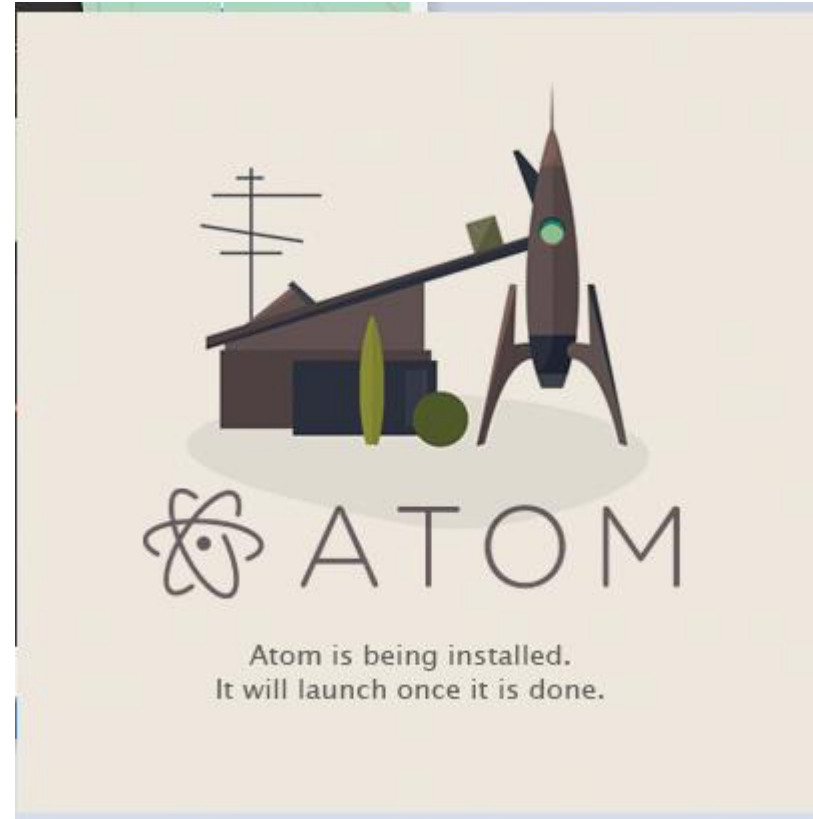
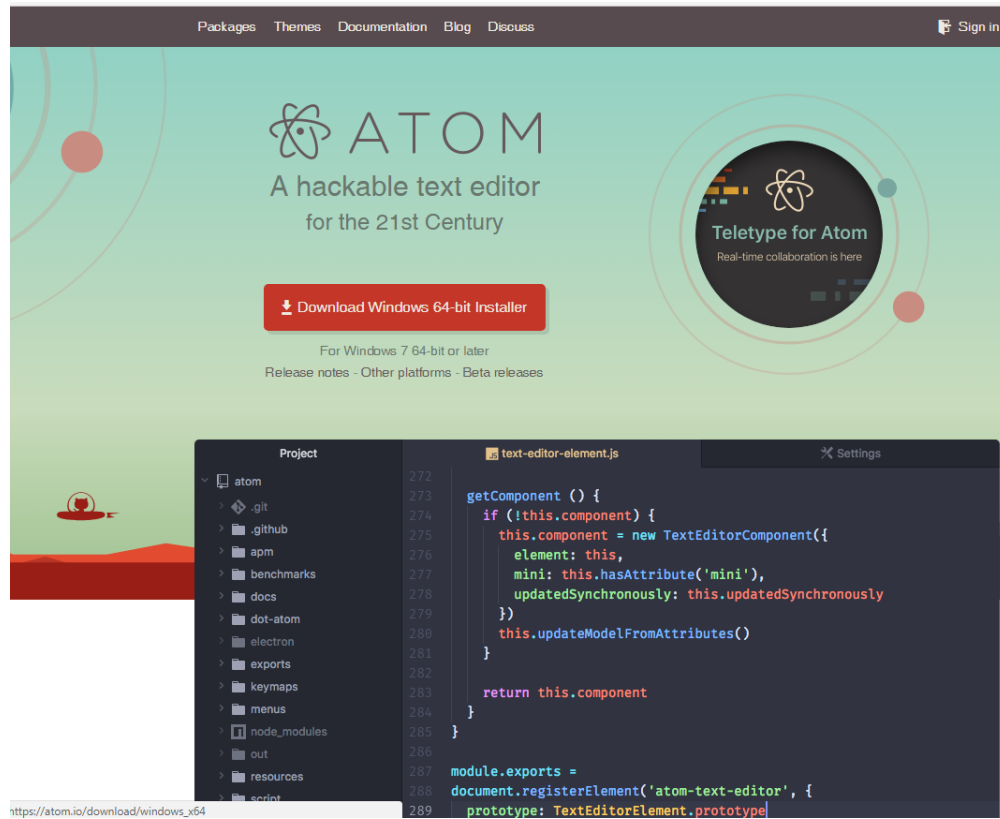
**Note that Python 3.5+ *cannot* be used on Windows XP or earlier.**

Not the OS you are looking for? Python can be used on many operating systems and environments.


[View the full list of downloads.](#)





# ATOM





## Get to know Atom!


 Open a Project


 Version control with Git and GitHub


 Collaborate in real time with Teletype


 Install a Package


 Choose a Theme


 Customize the Styling


 Hack on the Init Script


 Add a Snippet

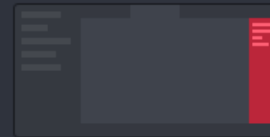
 Learn Keyboard Shortcuts

 Open a Project

 Version control with Git and GitHub

 Collaborate in real time with Teletype


 Install a Package



One of the best things about Atom is the package ecosystem. Installing packages adds new features and functionality you can use to make the editor suit your needs. Let's install one.

[Open Installer](#)

**Next time:** You can install new packages from the settings.

 Choose a Theme






## + Install Packages

? Packages are published to [atom.io](https://atom.io) and are installed to C:\Users\Funam\.atom\packages

Packages Themes

atom-runner 2.7.1

Runs scripts inside of Atom.


 Isegal

234.123

Install

script-runner 2.2.2

Run scripts and shells within a proper terminal.


 ioquatix

38.432

Install

grunt-runner 0.14.0

Run your grunt tasks from Atom.


 kokarn

28.121

Install

mocha-test-runner 1.0.0

Run Mocha tests from within Atom

 TabDigital


12.053

Install

Packages Themes

activate-power-mode 2.7.0

Activate POWER MODE to write your code in style.

 JoelBesada

730.115

Install


## + Install Packages

? Packages are published to [atom.io](https://atom.io) and are installed to C:\Users\Funam\.atom\packages

Packages Themes

atom-python-run 0.9.6

Run a python source file.

 foreshadow

154.961

Settings

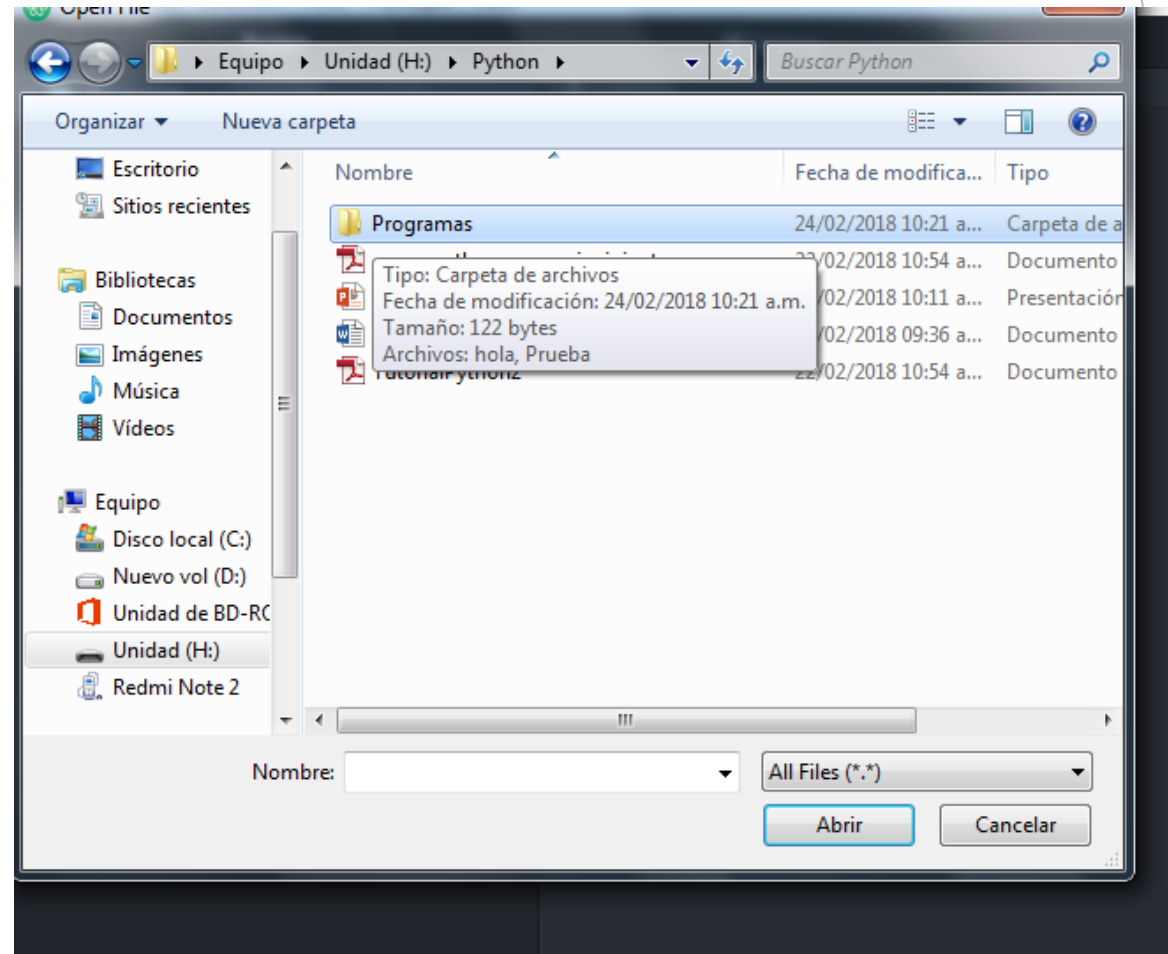
Uninstall

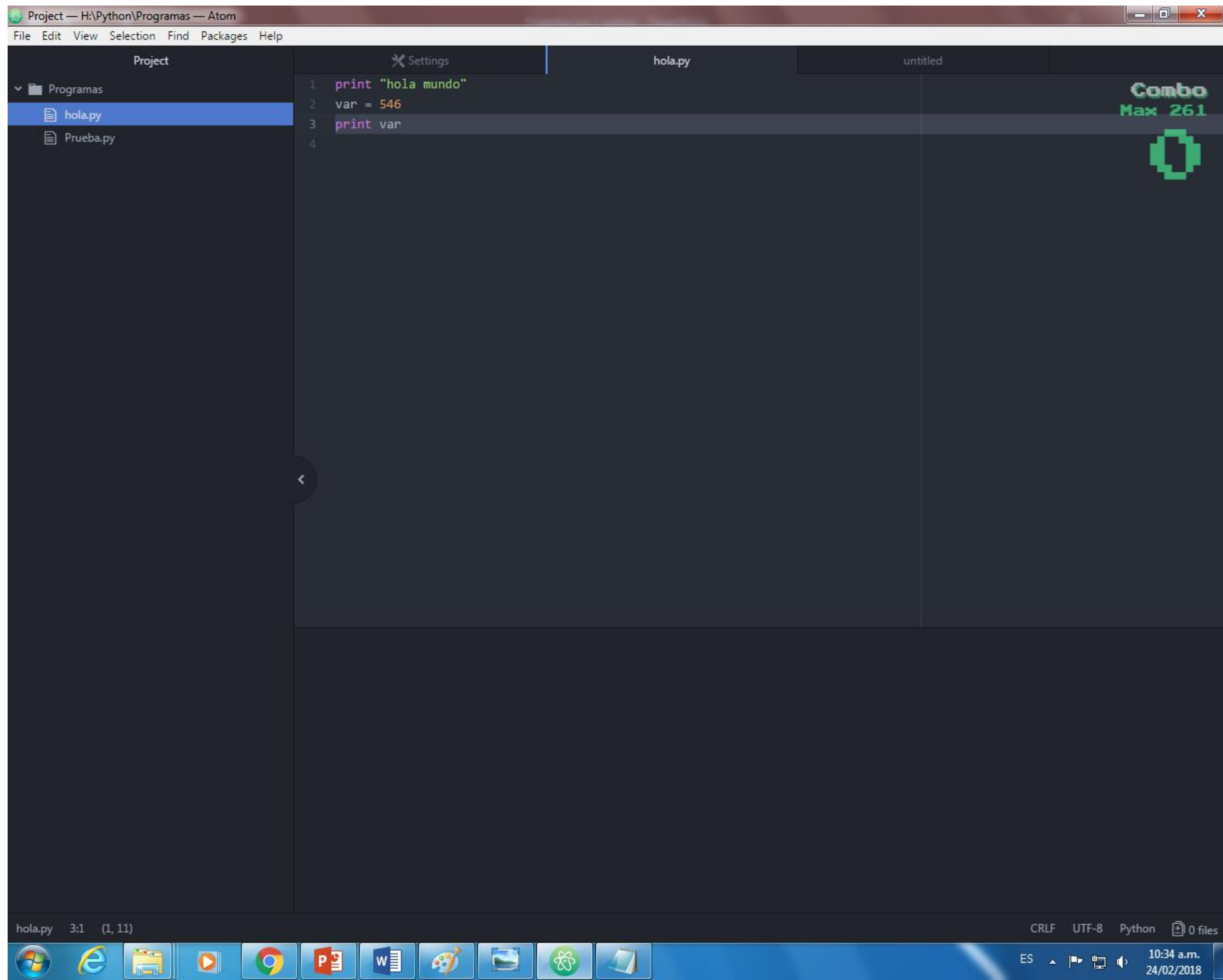
Disable

Crear un carpeta donde se guardara la información nuestros programas

## Atom

File	Edit	View	Selection	Find	Package
New Window	Ctrl+Shift+N				
New File	Ctrl+N				
Open File...	Ctrl+O				
Open Folder...	Ctrl+Shift+O				
Add Project Folder...	Ctrl+Shift+A				
Reopen Project					
Reopen Last Item	Ctrl+Shift+T				
Settings	Ctrl+Comma				
Config					





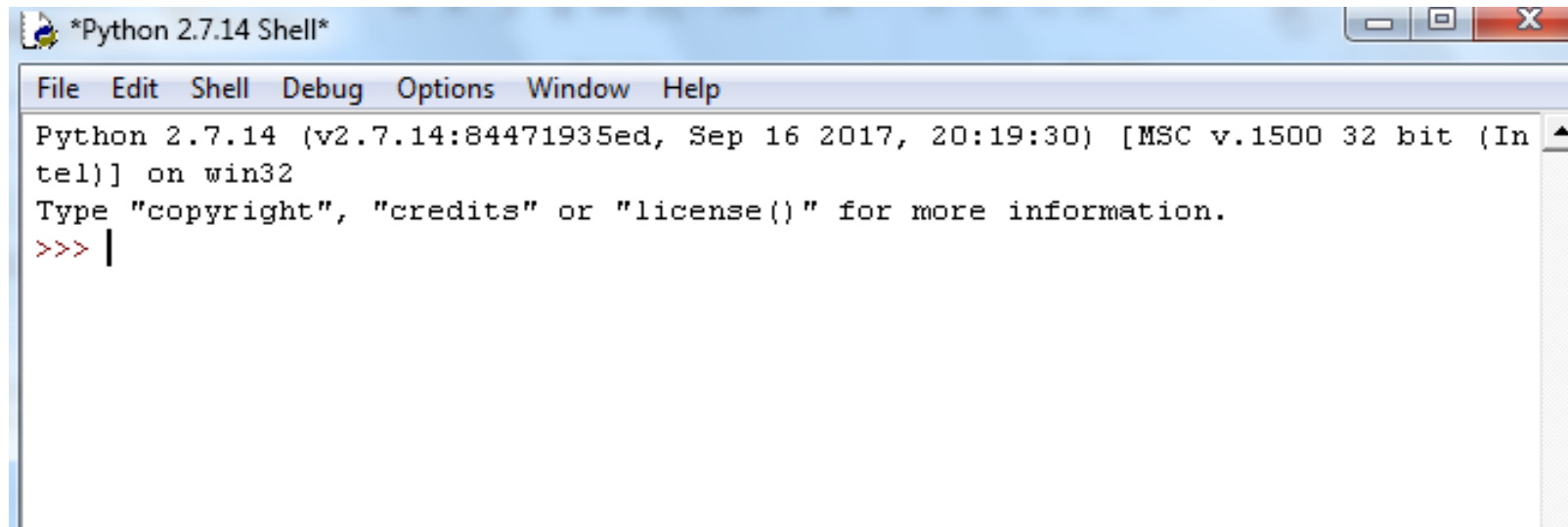
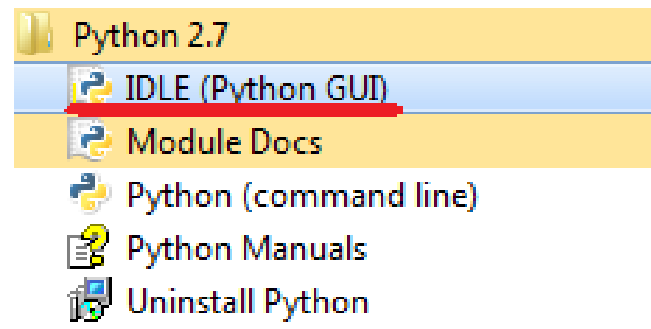
# Palabras reservadas en Python

- En los lenguajes informáticos, una palabra reservada es una palabra que tiene un significado gramatical especial para ese lenguaje y no puede ser utilizada como un identificador de objetos en códigos del mismo, como pueden ser las variables.

Son palabras reservadas en Python:

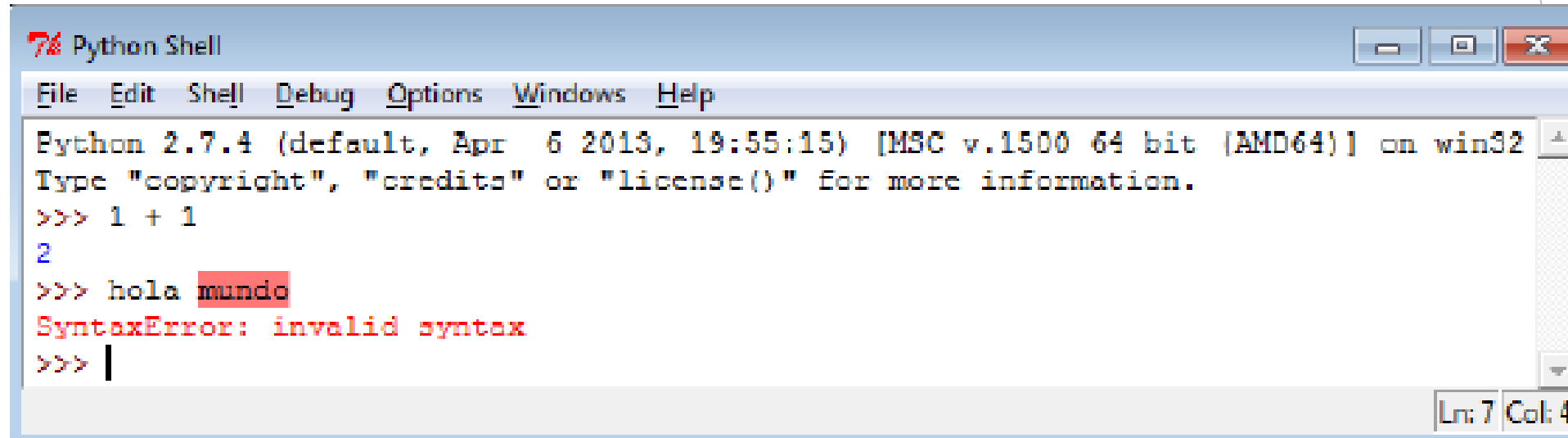
and	except	is
elif	import	return
global	print	def
or	class	for
assert	exec	while
else	in	try
if	raise	del
pass	continue	from
break	finally	not

# Primer Programa en Python desde el interprete o shell

A screenshot of the Python 2.7.14 Shell window. The window has a title bar that says '\*Python 2.7.14 Shell\*'. Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window contains the following text:

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

- Python cuenta con un intérprete de comandos llamado *Shell* o simplemente una terminal, la cual sirve para que ejecute las instrucciones que se ingresan de forma inmediata.



The screenshot shows a window titled "Python Shell" with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The text area contains the following text:

```
Python 2.7.4 (default, Apr 6 2013, 19:55:15) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 1 + 1
2
>>> hola mundo
SyntaxError: invalid syntax
>>> |
```

The status bar at the bottom right indicates "Ln: 7 Col: 4".

Python 2.7.14 Shell

File Edit Shell Debug Options Window Help

Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

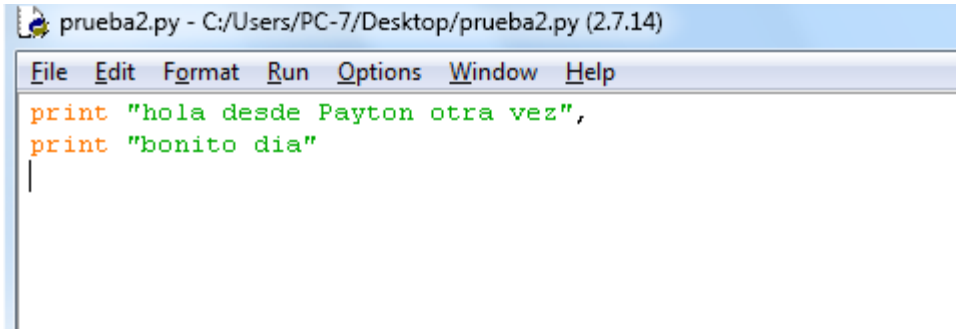
```
>>> print "hola desde Python"
```

```
hola desde Python
```

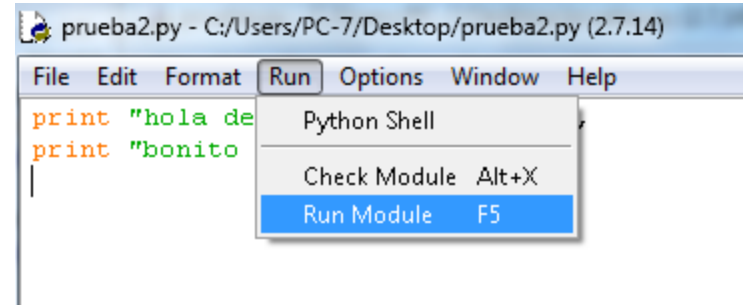
```
>>> |
```



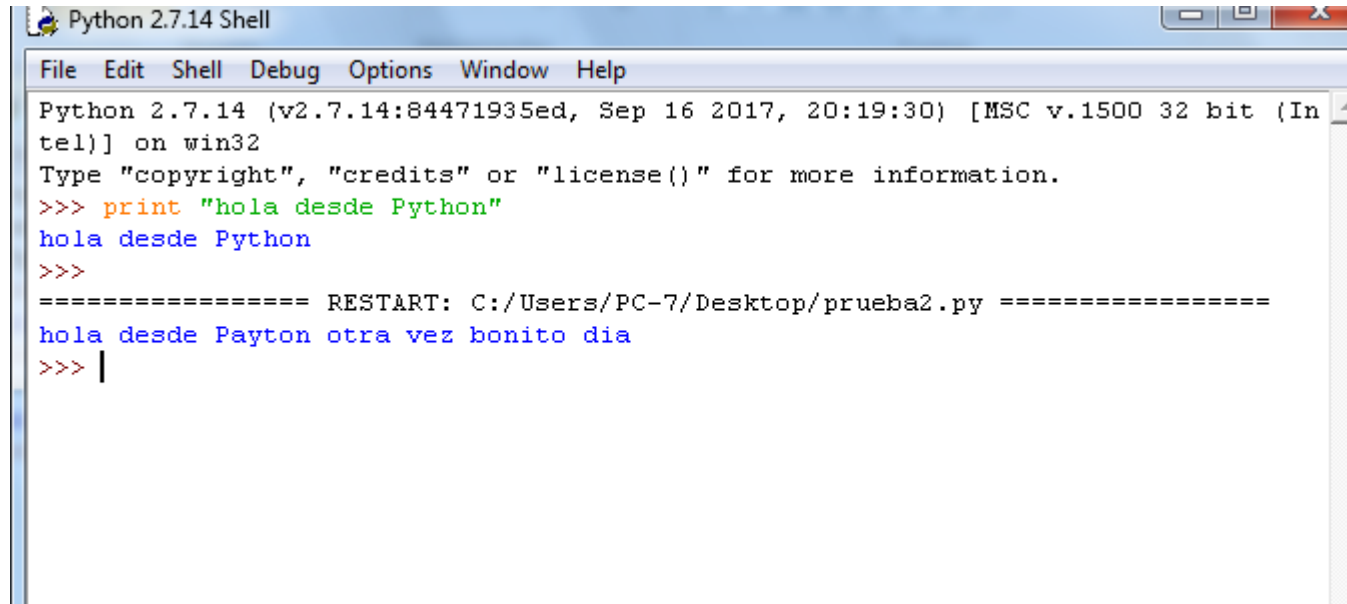
## Saludo desde un script de Python



```
prueba2.py - C:/Users/PC-7/Desktop/prueba2.py (2.7.14)  
File Edit Format Run Options Window Help  
print "hola desde Payton otra vez",  
print "bonito dia"
```



```
prueba2.py - C:/Users/PC-7/Desktop/prueba2.py (2.7.14)  
File Edit Format Run Options Window Help  
print "hola de  
print "bonito  
|  
Python Shell  
Check Module Alt+X  
Run Module F5
```



```
Python 2.7.14 Shell  
File Edit Shell Debug Options Window Help  
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> print "hola desde Python"  
hola desde Python  
>>>  
===== RESTART: C:/Users/PC-7/Desktop/prueba2.py =====  
hola desde Payton otra vez bonito dia  
>>> |
```

# Entrada y salida.

## Salida por pantalla: la función print()

- En Informática, la "salida" de un programa son los datos que el programa proporciona al exterior. Aunque en los inicios de la informática la salida más habitual era una impresora, hace muchos años que el dispositivo de salida más habitual es la pantalla del ordenador.

- ▶ **Salida por pantalla en IDLE**
- ▶ Para que IDLE muestre el valor de una variable, basta con escribir su nombre.

```
>>> a = 2
>>> a
2
```

- ▶ También se puede conocer el valor de varias variables a la vez escribiéndolas entre comas (IDLE las mostrará entre paréntesis), como muestra el siguiente ejemplo:

```
>>> a = b = 2
>>> c = "pepe"
>>> a
2
>>> c, b
('pepe', 2)
```

- ▶ La función `print()`
- ▶ En los programa, para mostrar texto o variables hay que utilizar la función `print()`.
- ▶ La función `print()` permite mostrar texto en pantalla. El texto a mostrar se escribe como argumento de la función:

```
print("Hola")
```

Hola

Las cadenas se pueden delimitar tanto por comillas dobles (") como por comillas simples (').

```
print('Hola')
```

Hola

# Entrada de Datos.

En Informática, la "entrada" de un programa son los datos que llegan al programa desde el exterior. Actualmente, el origen más habitual es el teclado.

- ▶ La función `input()`
- ▶ La función `input()` permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Intro, como muestra el siguiente ejemplo:

```
print("¿Cómo se llama?")
nombre = input()
print(f"Me alegro de conocerle,
{nombre}")
```

```
¿Cómo se llama?
Pepe
Me alegro de conocerle, Pepe
```

# Comentario en Python

- Los comentarios en Python, al igual que sucede en otros lenguajes de programación, sirven para explicar a las personas que puedan leer el programa en el futuro, qué es lo que hace el programa, así como explicar algunas partes del código. Estos comentarios son ignorados por las computadoras cuando ejecutan el código.

```
1 # Este es un comentario en python
2 >>> comando_python1
3 >>> comando_python2 # Esto también es un comentario
4
5 # Este es otro comentario en python
6 # aunque lo hemos escrito en dos líneas
7
8 """ Este es un comentario multilinea. La
9 siguiente parte realiza una serie
10 de cosas muy chulas """
11 >>> comando_chulo_python1
12 >>> comando_chulo_python2
13
14 """ Aunque es multilinea, sólo usamos una """
```

[\*] main.cpp

```
1 #include <cstdlib>
2 #include <iostream>
3
4 using namespace std;
5
6 int main(int argc, char *argv[])
7 {
8     system("PAUSE");
9     return EXIT_SUCCESS;
10 }
11
```

Sin Nombre1 [\*] practica1.cpp

```
1 // directivas de preprocesador
2
3 #include <iostream.h>
4 #include <cstdlib>
5
6 //declaraciones globales
7 using namespace std;
8
9 int main() //funcion principal
10 { //inicia bloque main
11
12     cout <<"hola mundo"<<endl;
13
14     system("pause");
15     return 0;
16 } //termina bloque main
```

HolaMundo.java (~/prueba) - gedit

Archivo Editar Ver Buscar Herramientas Documentos Ayuda

Nuevo Abrir Guardar Imprimir... Deshacer Rehacer Cortar Copiar

HolaMundo.java

```
public class HolaMundo {
    /**
     * Primera aplicación Java
     */
    public static void main (String[] args) {
        // Muestra en la consola el texto Hola Mundo
        System.out.println("Hola Mundo");
    }
}
```

java Ancho de la tabulación: 8 Ln 9, Col 1 INS

# Operadores con Python

## (Operadores +, -, \* y /)

```
>>> 2+2
```

```
4
```

```
>>> # Este es un comentario
```

```
... 2+2
```

```
4
```

```
>>> 2+2 # y un comentario en la misma línea que el código
```

```
4
```

```
>>> (50-5*6)/4
```

```
5
```



# Uso de Variables

- ▶ En programación, las **variables** son espacios reservados en la memoria que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa. Una variable corresponde a un área reservada en la memoria principal del ordenador.
- ▶ En Python: Las variables en Python se definen de forma dinámica, lo que significa que no se tiene que especificar cuál es su tipo. Para crear una variable se escribe una igualdad con el nombre de la variable a la izquierda y el valor que se quiere asignar a la derecha.

**nombre\_de\_la\_variable = valor\_de\_la\_variable**

- El nombre de una variable debe ser nemotécnico, esto quiere decir que hay una relación entre el nombre y el dato guardado en la variable, haciendo más sencillo entender qué representa una variable al leer una porción de código. También debe comenzar por una letra o un guión bajo y no pueden ser palabras reservadas.

```
>>> nombre="Aaron Velasco"  
>>> edad=24  
>>> facultad="Aragon"  
>>> estatura=1.79  
>>> peso=78
```

Ejercicio: declarar variables con sus fatos personales (7 campos)

- El signo igual (=) es usado para asignar un valor a una variable. Luego, ningún resultado es mostrado.

```
>>> ancho = 20
>>> largo = 5*9
>>> ancho * largo
900
```

Hacer: suma , resta,  
multiplicación, división, modulo

Hacer: sacar el área de un  
Triangulo

- Las variables deben estar "definidas" (con un valor asignado) antes de que puedan usarse, o un error ocurrirá:

```
>>> n
```

```
Traceback (most recent call last):  
  File "<pyshell#15>", line 1, in <module>  
    n  
NameError: name 'n' is not defined
```

- ▶ La función print()
- ▶ En los programa, para mostrar texto o variables hay que utilizar la función print().
- ▶ La función print() permite mostrar texto en pantalla. El texto a mostrar se escribe como argumento de la función:

```
iNumero=10  
palabra="hola"  
fNumero=10.05  
cadena="esta es una cadena de palabras"
```

```
print iNumero  
print "este es el valor de iNumero", iNumero
```

Ejercicio: imprimir varias variables.

# La función input()

- La función input() permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla **Intro**.

```
print("¿Cómo se llama? ", end="")  
nombre = input()  
print(f"Me alegro de conocerle, {nombre}")
```

```
print("Hola")
print ("Dame datos:")

nombre=input("Nombre :")

print("Me da gusto conocerte: ",nombre)

apellidol=input("Paterno: ")
edad=input("Edad : ")

print("Tus datos son: ")
print ("Nombre: ",nombre," Apellido: ",apellidol," Edad: ",edad)

input("Precione cual quier tecla para continuar...")
```



- De forma predeterminada, la función `input()` convierte la entrada en una cadena, aunque escribamos un número. Si intentamos hacer operaciones, se producirá un error.

```
print("Suma de numeros")
num1=input("Dame un numero: ")
num2=input("Dame otro numero")

resultado=num1+num2
print("El resultado de la suma es",resultado)

input("Preciona cualquier tecla para continuar")
|
```

```
num1=int (input("Dame un numero: "))
```

```
num2=int (input("Dame otro numero"))
```

```
resultado=num1+num2
```

```
|
```

```
print ("Calculo del Iva de una cantidad")

print("1.- Digita el iva")
print("2.- Digita la catidad")

ivacli=float(input("Dame el iva: "))

cantidad=float(input("Dame una catidad: "))

print("La catidad es: ",cantidad)
print("El Iva es: ",ivacli*cantidad)

input("Preciona cualquier tecla para continuar....")
```

# Tipos de datos

- ▶ Cadena de texto (string):
- ▶ `mi_cadena = "Hola Mundo!"`

`mi_cadena_multilinea = """`

`Esta es una cadena  
de varias líneas"""`

```
1  cadena = """esta es una ceda
2  de muchas lineas demacias """
3  cadena2="cadena normal"
4
5  print cadena
6
7  print cadena2
8
```

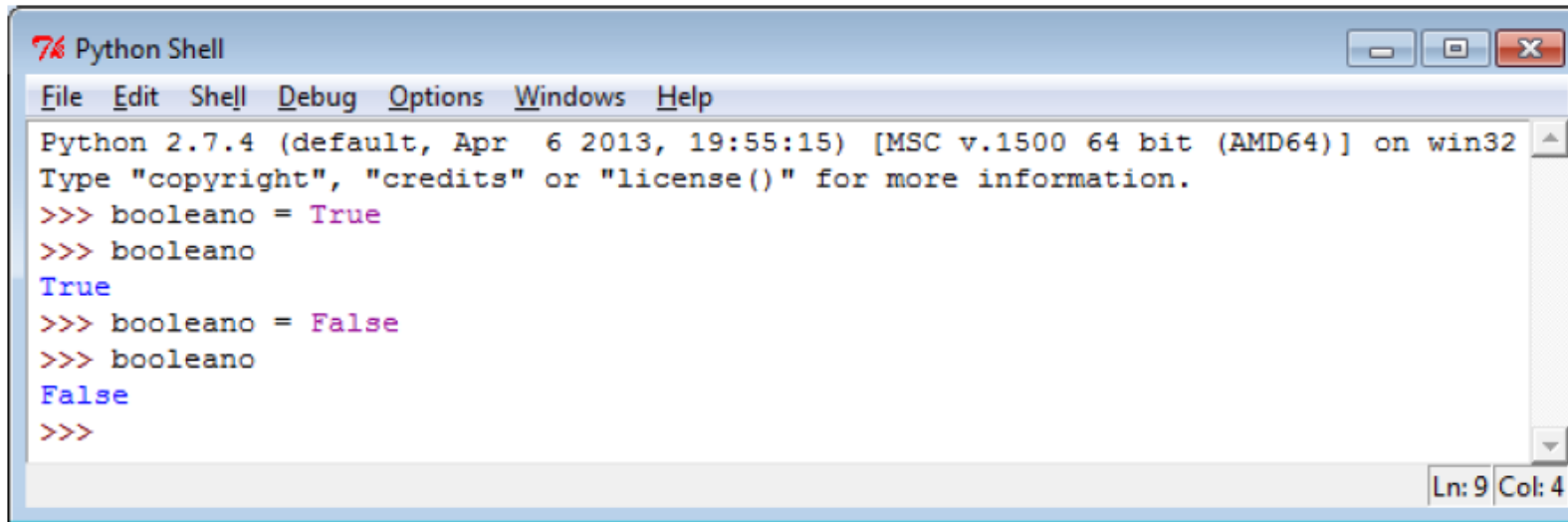
C:\Python27\python.exe

```
esta es una ceda
de muchas lineas demacias
cadena normal
```

# Tipos de datos

## ► Booleano (*bool*)

- El tipo de dato booleano (bool de boolean) es aquel que sólo puede tener dos valores: True (verdadero) o False (falso). Como ya se mencionó, Python no necesita tener especificado el tipo de dato cuando se declara.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.4 (default, Apr 6 2013, 19:55:15) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> booleano = True
>>> booleano
True
>>> booleano = False
>>> booleano
False
>>>
```

Ln: 9 Col: 4

Para ver qué tipo de dato es una variable, se utiliza la sintaxis ***type(objeto)***

# Entero (*int*)

- El tipo de dato entero (*int* de *integer*) resulta ser aquel que puede tomar valores enteros desde -2147483648 hasta 2147483647. Si el valor sale de estos límites, Python lo volverá un dato largo (*long*), el cual dependerá de que la memoria de la computadora no se acabe.

```
>>> entero = 123
>>> entero
123
>>> type(entero)
<type 'int'>
>>> entero = 2147483648
>>> entero
2147483648L
>>> type(entero)
<type 'long'>
>>>
```

# Casteo: es convertir un tipo de dato a otro:

- ▶ Entero -> Booleano.
- ▶ Cadena -> Entero.
- ▶ Entero -> Cadena.
- ▶ Booleano -> Cadena.
- ▶ Para “castear”, se necesita encerrar el objeto que se quiera convertir en la clase del nuevo tipo de objeto a convertir, con la sintaxis

▶ **claseTipoDato(objeto)**



# Ejemplo de casteo

```
>>> booleano = True
>>> entero = int(booleano)
>>> entero
1
>>> entero = 20
>>> entero
20
>>> booleano = bool(entero)
>>> booleano
True
...
```

# Flotante (*float*)

- El tipo de dato flotante (`float`) es aquel que puede tomar valores reales o decimales, teniendo una precisión de 53 dígitos. Este tipo de datos es fácilmente diferenciado por tener un punto decimal.

```
>>> flotante = 1.23456789
>>> flotante
1.23456789
>>> type(flotante)
<type 'float'>
>>>
```

---

# Cadena (*str*) -> String

- El tipo de dato cadena (*str* de *string*) es un conjunto de caracteres. Se declaran con comillas simples o comillas dobles, y la máxima longitud dependerá del número de bits del Sistema Operativo, es decir, cerca de 63 GB para sistemas de 64 bits y alrededor de dos a tres GB en sistemas de 32 bits

```
>>> cadena = "Una cadena"
>>> cadena
'Una cadena'
>>> type(cadena)
<type 'str'>
>>> cadena2 = 'Otra cadena'
>>> cadena2
'Otra cadena'
>>> type(cadena2)
<type 'str'>
>>>
```

# Cadena de caracteres

- Debido a que en realidad no existe el tipo de dato caracter (char de character) en Python, se necesitan manejar a las cadenas como una lista que tiene un índice numérico. Esto se logra con la sintaxis cadena[índice]

```
>>> cadena = "Hola"
>>> cadena[0]
'H'
>>> cadena[3]
'a'
>>> cadena[4]

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    cadena[4]
IndexError: string index out of range
>>> letra = cadena[0]
>>> letra
'H'
>>> type(letra)
<type 'str'>
>>>
```

Para conocer la longitud de un objeto, se utiliza la instrucción ***len(objeto)***

## Concatenar:

Este término significa juntar cadenas de caracteres. El proceso de concatenación se realiza mediante el operador de suma (+). Ten en cuenta que debes marcar explícitamente dónde quieres los espacios en blanco y colocarlos entre comillas.

```
mensaje1 = 'Hola' + ' ' + 'Mundo'  
print(mensaje1)  
-> Hola Mundo
```

- ▶ Multiplicar
- ▶ Si quieres varias copias de una cadena de caracteres utiliza el operador de multiplicación (\*). En este ejemplo, la cadena de caracteres mensaje2a lleva el contenido “Hola” tres veces, mientras que la cadena de caracteres mensaje2b tiene el contenido “Mundo”. Ordenemos imprimir las dos cadenas.

```
mensaje2a = 'Hola ' * 3
mensaje2b = 'Mundo'
print(mensaje2a + mensaje2b) -> Hola Hola Hola
Mundo
```

# Minúsculas

- A veces es útil convertir una cadena de caracteres a minúsculas. Para ello se utiliza el método `lower`. Por ejemplo, al uniformar los caracteres permitimos que la computadora reconozca fácilmente que “Algunas Veces” y “algunas veces” son la misma frase.

```
mensaje7 = "HOLA MUNDO"  
mensaje7a = mensaje7.lower()  
print(mensaje7a)  
-> hola mundo
```

# Cadenas (Listas)

- ▶ En algunos lenguajes de programación se las conocen como arreglos o matrices; y se caracterizan porque los elementos están entre corchetes y separados por una coma.
- ▶ ¿Qué es una lista?
- ▶ Una lista es una estructura de datos y un tipo de dato en python con características especiales. Lo especial de las listas en Python es que nos permiten almacenar cualquier tipo de valor como enteros, cadenas y hasta otras funciones;



+---+---+---+---+---+---+  
| P | y | t | h | o | n |  
+---+---+---+---+---+---+  
0 1 2 3 4 5 6  
-6 -5 -4 -3 -2 -1

```
valor= True
id_aaron="413112576-a"

lista = [1,23,"aaron",1.75,id_aaron,"velasco",valor]

print lista
```

El tipo de dato lista tiene algunos métodos

```
valor= True
```

```
id_aaron="413112576-a"
```

```
lista = [1,23,"aaron",1.75,id_aaron,"velasco",valor]
```

```
tipo_lista = type(lista)
```

```
tamano_lista = len(lista)
```

```
print tamano_lista
```

**count(objeto):** Devuelve cuántos objetos hay en la lista.

`nombre_lista.count (objeto)`

**index(objeto):** Devuelve en qué índice se encuentra el objeto enviado.

`nombre_lista.index (objeto)`

***append(objeto):*** Agrega el **objeto** a la lista hasta el final de la lista.

`nombre_lista.append (“objeto o parámetro a agregar”)`

***extend(secuencia):*** Parecido al método *append*, pero aquí se recibe como parámetro una **secuencia**, como puede ser una cadena o una estructura de datos, y la añade al final de la lista.

`nombre_lista.index (“secuencia de parámetros a agregar ”)`

```
>>> lista.extend((1, 2, 3))
>>> lista
['Primero', 2, 3.1416, 'Cuarto', 1, 2, 3]
>>> lista.extend("Cadena")
>>> lista
['Primero', 2, 3.1416, 'Cuarto', 1, 2, 3, 'C', 'a', 'd', 'e', 'n', 'a']
```

***insert(índice, objeto)***: Añade el **objeto** a la lista en la posición **índice** que se indica. Si el **índice** es más grande que la longitud de la lista, se agregará al objeto hasta el final de la lista.

```
>>> lista.insert(0, 0)
>>> lista
[0, 'Primero', 2, 3.1416, 'Cuarto', 1, 2, 3, 'C', 'a', 'd', 'e', 'n', 'a']
>>> lista.insert(1000, 1000)
>>> lista
[0, 'Primero', 2, 3.1416, 'Cuarto', 1, 2, 3, 'C', 'a', 'd', 'e', 'n', 'a', 1000]
```

***pop(índice)***: Devuelve el elemento del **índice** establecido y lo elimina de la lista. Este parámetro es **opcional**, por lo que, si se opta por colocarlo, se eliminará el último elemento de la lista. Si el índice rebasa la longitud de la lista, se producirá un error.

```
>>> lista.pop()
1000
>>> lista
[0, 'Primero', 2, 3.1416, 'Cuarto', 1, 2, 3, 'C', 'a', 'd', 'e', 'n', 'a']
```

***remove(objeto)***: Retira de la lista el **objeto** indicado. Se elimina el primer objeto localizado en la lista, por lo que, si no está en la lista, se producirá un error indicándolo.

```
>>> lista.remove(0)
>>> lista
[2, 3.1416, 'Cuarto', 1, 2, 3, 'C', 'a', 'd', 'e', 'n', 'a']
```

***reverse()***: Invierte el orden de los datos que tiene la lista, siendo el primero el último, el segundo el penúltimo y así sucesivamente.

```
>>> lista.reverse()  
>>> lista  
['a', 'n', 'e', 'd', 'a', 'C', 3, 2, 1, 'Cuarto', 3.1416, 2]
```

***sort()***: Ordena los datos que contiene la lista de acuerdo al tipo de datos.

```
>>> lista.sort()  
>>> lista  
[1, 2, 2, 3, 3.1416, 'C', 'Cuarto', 'a', 'a', 'd', 'e', 'n']
```

# Tipos de operadores

- ▶ Los operadores son útiles para realizar diferentes objetivos:
- ▶ Operadores de comparación, como su nombre lo indica, permiten comparar dos valores.
- ▶ Operaciones aritméticas básicas, como lo son la suma, resta, multiplicación y división.
- ▶ Comparaciones lógicas, agrupa varias condiciones que se deben o no cumplir. La evaluación se realiza en forma booleana, esto es, verdadero o falso.

# Comparativos

- Son aquellos que permiten comparar dos tipos de dato y devolver un resultado booleano.

Operador	Significado
<	Mayor que
>	Menor que
==	Igual que
!=	Diferente que
>=	Mayor o igual que
<=	Menor o igual que

Estos operadores siempre se colocan entre los dos datos que se comparen, no al inicio ni final.

```
>>> 1 > 2
False
>>> 3 < 1
False
>>> 5 == 5
True
>>> 2 != 4
True
>>> 3 >= 3
True
>>> 2 <= 3
True
>>>
```



# Lógicos

- ▶ Son aquellos que adjuntan una serie de valores booleanos, como pueden ser evaluar condiciones, para devolver un valor booleano final que dependerá de su lógica. Existen tres comparadores lógicos en total:
- ▶ 1. **and**: Hará que la condición final devuelva **True** siempre y cuando todos los valores también resulten **True**. Si uno de ellos resulta **False**, se devuelve el valor **False**.

Operador 1	Operador 2	Resultado
<i>False</i>	<i>False</i>	<i>False</i>
<i>False</i>	<i>True</i>	<i>False</i>
<i>True</i>	<i>False</i>	<i>False</i>
<i>True</i>	<i>True</i>	<i>True</i>

```
>>> 3 == 4 and 2 < 1
False
>>> 3 != 4 and 2 < 1
False
>>> 3 == 4 and 2 > 1
False
>>> 3 != 4 and 2 > 1
True
>>>
```

- 2. **or**: Permite que la condición final devuelva **True** cuando al menos uno de los valores sea **True**. Si todos resultan ser **False**, se devuelve el valor **False** al final. En la figura 1.18 se muestra la tabla del operador **or**.

Operador 1	Operador 2	Resultado
<i>False</i>	<i>False</i>	<i>False</i>
<i>False</i>	<i>True</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>True</i>
<i>True</i>	<i>True</i>	<i>True</i>

```
>>> 1 > 2 or 123 > 456
False
>>> 1 < 2 or 123 > 456
True
>>> 1 > 2 or 123 < 456
True
>>> 1 < 2 or 123 < 456
True
>>> True or True or False
True
>>> False or False or False
False
```

- 3. **not**: Devuelve el valor opuesto al ingresado, es decir, si el valor resulta ser **True**, se devuelve **False** al final y viceversa. En la figura 1.20 se puede apreciar una tabla con el funcionamiento de este operador.

Operador 1	Operador 2	Resultado
<i>False</i>	<i>False</i>	<i>False</i>
<i>False</i>	<i>True</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>True</i>
<i>True</i>	<i>True</i>	<i>True</i>

```
>>> 1 > 2 or 123 > 456
False
>>> 1 < 2 or 123 > 456
True
>>> 1 > 2 or 123 < 456
True
>>> 1 < 2 or 123 < 456
True
>>> True or True or False
True
>>> False or False or False
False
```

# Aritméticos

- ▶ Son todos los que permiten realizar operaciones entre los tipos de datos. Las cuatro operaciones más básicas son:
  - ▶ 1. Suma (+).
  - ▶ 2. Resta (-).
  - ▶ 3. Multiplicación (\*).
  - ▶ 4. División (/).
- ▶ Si se dividen dos números enteros, el resultado será un entero también, aunque el resultado sea un número decimal. Para solucionar este problema, al menos uno de los divisores debe ser flotante para obtener al final un flotante también.

Símbolo	Significado	Ejemplo	Resultado
+	Suma	<code>a = 10 + 5</code>	<code>a es 15</code>
-	Resta	<code>a = 12 - 7</code>	<code>a es 5</code>
-	Negación	<code>a = -5</code>	<code>a es -5</code>
*	Multiplicación	<code>a = 7 * 5</code>	<code>a es 35</code>
**	Exponente	<code>a = 2 ** 3</code>	<code>a es 8</code>
/	División	<code>a = 12.5 / 2</code>	<code>a es 6.25</code>
//	División entera	<code>a = 12.5 / 2</code>	<code>a es 6.0</code>
%	Módulo	<code>a = 27 % 4</code>	<code>a es 3</code>

# Precedencia de operadores

- Al final de esto, se debe tener en cuenta que todos los lenguajes de programación manejan una jerarquía hacia todos los operadores, y Python no es la excepción. Si se quiere dar prioridad a alguna operación, se deberá encerrar entre paréntesis para realizarse antes de otro operador con mayor jerarquía.

```
>>> 3 + 4 * 5
23
>>> (3 + 4) * 5
35
>>> 3 + (4 * 5)
23
>>> 4 == 5 and 5 != 0 or 1 > 0
True
>>> (4 == 5 and 5 != 0) or 1 > 0
True
>>> 4 == 5 and (5 != 0 or 1 > 0)
False
>>>
```

Operador	Descripción
(, )	Paréntesis
**	Potencia
*, /, %, //	Multiplicación, división, módulo y cociente
+, -	Adición y substracción
<=, <, >, >=	Operadores comparativos
==, !=	Operadores de igualdad
and, or, not	Operadores lógicos

# Estructuras de Control.

## Primeros pasos hacia la programación.

- Tres tipos de Estructuras:



• *Secuencial*

• *Instrucción condicional.*

• *Iteración (bucle de instrucciones)*

# Condicional, if ... elif ... else...

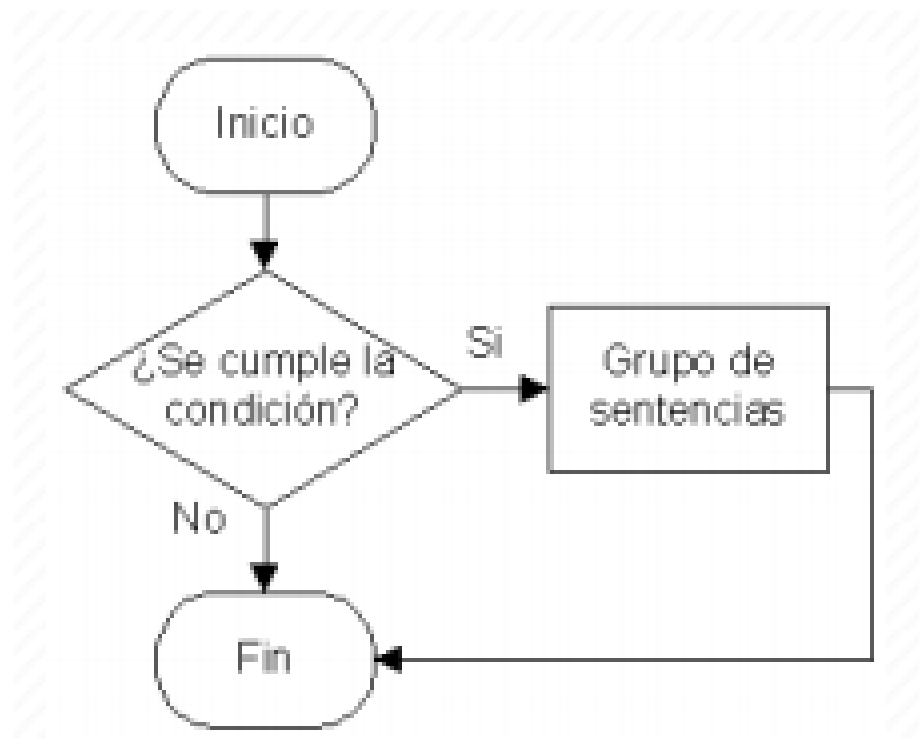
- ▶ **Sentencias condicionales: if ...**
- ▶ La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición. En inglés "if" significa "si" (condición).

```
if condición:  
    aquí van las órdenes que se ejecutan si la condición es cierta  
    y que pueden ocupar varias líneas
```

La primera línea contiene la condición a evaluar y es una expresión lógica. Esta línea debe terminar siempre por dos puntos (:).

A continuación viene el bloque de órdenes que se ejecutan cuando la condición se cumple (es decir, cuando la condición es verdadera). Es importante señalar que este bloque debe ir sangrado, puesto que Python utiliza el sangrado para reconocer las líneas que forman un bloque de instrucciones.





```
numero = int(input("Escriba un  
número positivo: "))  
if numero < 0:  
    print("¡Le he dicho que  
escriba un número positivo!")  
print(f"Ha escrito el número  
{numero}")
```

```
Escriba un número positivo: -5  
¡Le he dicho que escriba un  
número positivo!  
Ha escrito el número -5
```

# Bifurcaciones: if ... else ...

- La estructura de control if ... else ... permite que un programa ejecute unas instrucciones cuando se cumple una condición y otras instrucciones cuando no se cumple esa condición. En inglés "if" significa "si" (condición) y "else" significa "si no".

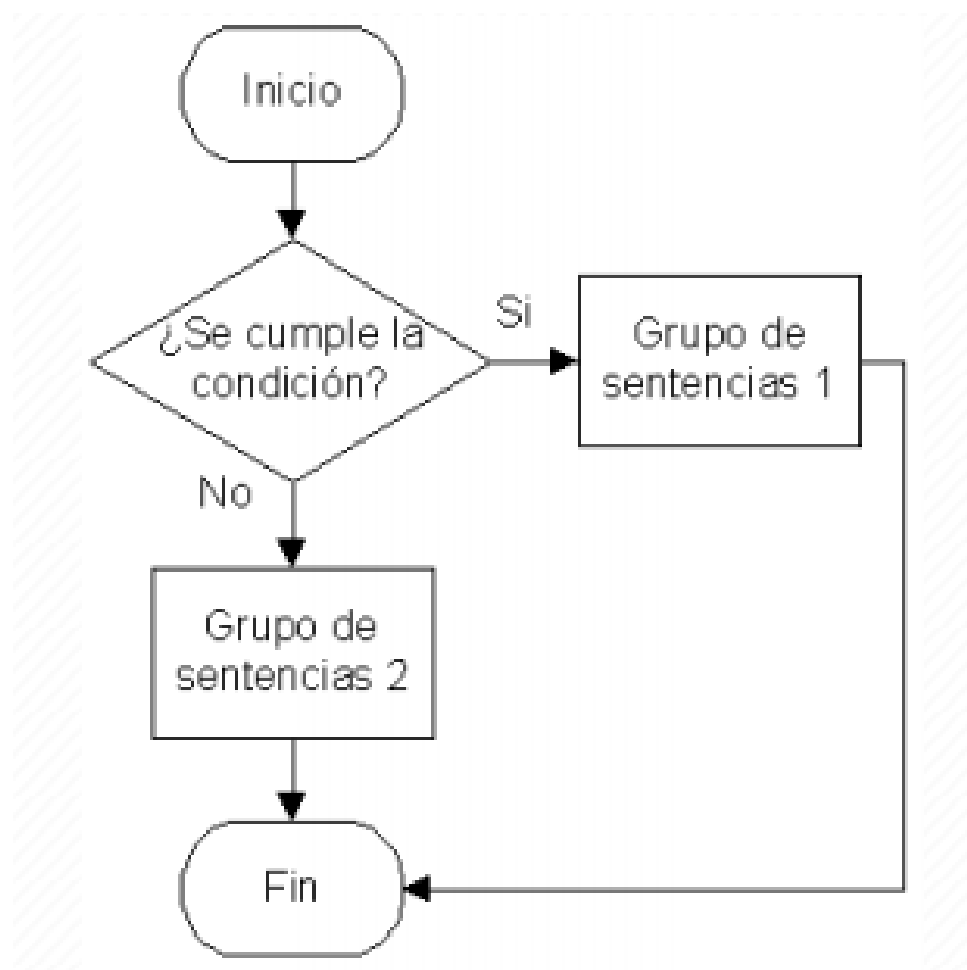
**if** condición:

aquí van las órdenes que se ejecutan si la condición es cierta  
y que pueden ocupar varias líneas

**else:**

y aquí van las órdenes que se ejecutan si la condición es  
falsa y que también pueden ocupar varias líneas

---



```
edad = int(input("¿Cuántos años  
tiene? "))  
if edad < 18:  
    print("Es usted menor de  
edad")  
else:  
    print("Es usted mayor de  
edad")  
print("¡Hasta la próxima!")
```

¿Cuántos años tiene? 17  
Es usted menor de edad  
¡Hasta la próxima!

```
edad = int(input("¿Cuántos años  
tiene? "))  
if edad < 18:  
    print("Es usted menor de  
edad")  
else:  
    print("Es usted mayor de  
edad")  
print("¡Hasta la próxima!")
```

¿Cuántos años tiene? 25  
Es usted mayor de edad  
¡Hasta la próxima!

- Sangrado de los bloques
- Un bloque de instrucciones puede contener varias instrucciones. Todas las instrucciones del bloque deben tener el mismo sangrado:

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
    print("Recuerde que está en la edad de aprender")
else:
    print("Es usted mayor de edad")
    print("Recuerde que debe seguir aprendiendo")
print("¡Hasta la próxima!")
```

# if ... elif ... else ...

- La estructura de control if ... elif ... else ... permite encadenar varias condiciones. elif es una contracción de else if.

<pre>if condición_1:     bloque 1 elif condición_2:     bloque 2 else:     bloque 3</pre>	<div>→</div> <div>→</div> <div>→</div>	<pre>if condición_1:     bloque 1 else:     if condición_2:         bloque 2     else:         bloque 3</pre>
---	--	---

- Si se cumple la condición 1, se ejecuta el bloque 1
- Si no se cumple la condición 1 pero sí que se cumple la condición 2, se ejecuta el bloque 2
- Si no se cumplen ni la condición 1 ni la condición 2, se ejecuta el bloque 3.

```
edad = int(input("¿Cuántos años tiene? "))
if edad >= 18:
    print("Es usted mayor de edad")
elif edad < 0:
    print("No se puede tener una edad negativa")
else:
    print("Es usted menor de edad")
```



# Iteración: Bucle for

- En general, un bucle es una estructura de control que repite un bloque de instrucciones. Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

```
for variable in elemento iterable (lista, cadena, range, etc.):  
    cuerpo del bucle
```

---



```
print("Comienzo")
for i in [0, 1, 2]:
    print("Hola ", end="")
print()
print("Final")
```

Comienzo  
Hola Hola Hola  
Final

```
print("Comienzo")
for i in []:
    print("Hola ", end="")
print()
print("Final")
```

Comienzo  
  
Final

- La función `range()`
- La función `range()` crea una lista de números enteros en sucesión aritmética. La función `range()` puede tener 1, 2 o 3 argumentos numéricos.

```
>>>range(5,10)
[5, 6, 7, 8, 9]
>>>range(-5,1)
[-5, -4, -3, -2, -1, 0]
>>>range(5,1)
[ ]
```

---

```
>>> # One parameter
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

```
>>> # Two parameters
>>> for i in range(3, 6):
...     print(i)
...
3
4
5
```

```
-
>>> # Three parameters
>>> for i in range(4, 10, 2):
...     print(i)
...
4
6
8
>>> # Going backwards
>>> for i in range(0, -10, -2):
...     print(i)
...
0
-2
-4
-6
-8
```

```
for i in "AMIGO":  
    print(f"Dame una {i}")  
print("¡AMIGO!")
```

Dame una A  
Dame una M  
Dame una I  
Dame una G  
Dame una O  
¡AMIGO!

```
print("Comienzo")  
for i in range(3):  
    print("Hola ", end="")  
print()  
print("Final")
```

Comienzo  
Hola Hola Hola  
Final

```
veces = int(input("¿Cuántas veces  
quiere que le salude? "))  
for i in range(veces):  
    print("Hola ", end="")  
print()  
print("Adios")
```

```
¿Cuántas veces quiere que le  
salude? 6  
Hola Hola Hola Hola Hola Hola  
Adios
```

## ► Contadores y acumuladores

- En muchos programas se necesitan variables que cuenten cuántas veces ha ocurrido algo (contadores) o que acumulen valores (acumuladores). Las situaciones pueden ser muy diversas, por lo que simplemente hay aquí un par de ejemplos para mostrar la idea.

```
for i in range (50,89):  
    print ("valor de I ->: ",i)  
print ("final")  
|
```

```
numero=3
```

```
for elemento in range (1,11):
```

```
    producto=numero*elemento
```

```
    print (producto)
```

```
print ("final")
```

```
numero = int (input ('Ingresa numero a multiplicar'))
```

```
##numero=3
```

```
for elemento in range (1,11):
```

```
    producto=numero*elemento
```

```
    print (producto)
```

```
print ("final")
```



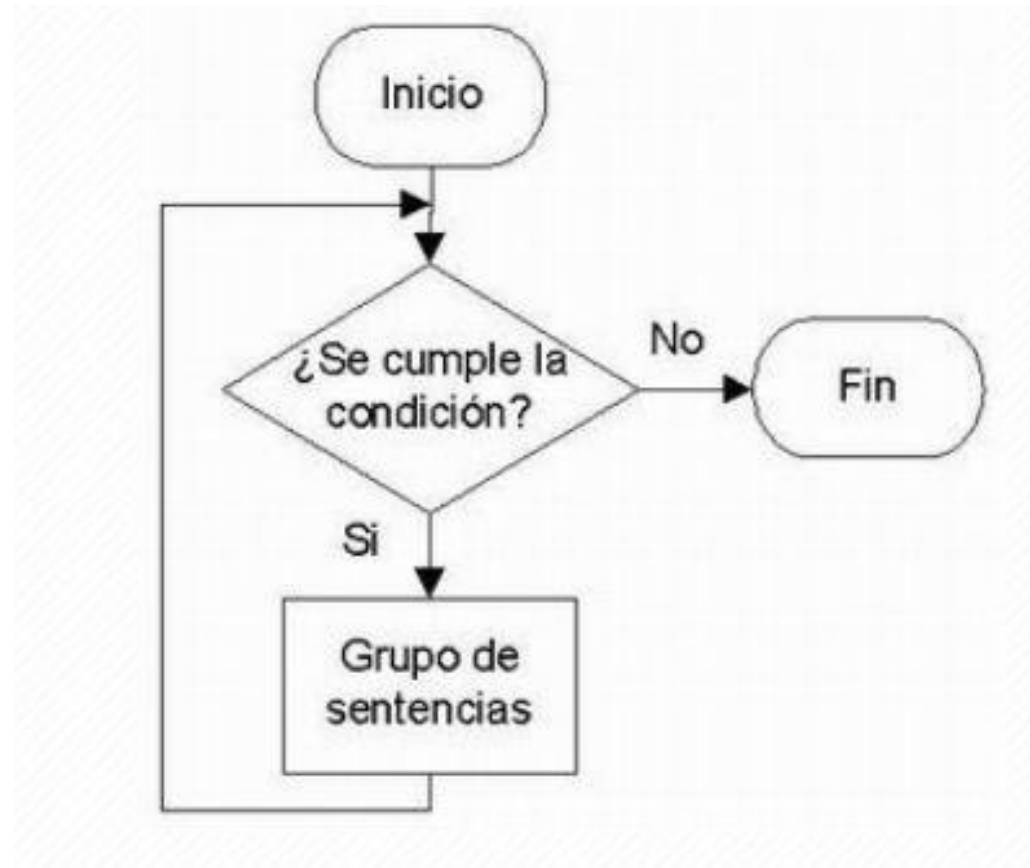
# While

- Un bucle while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True) .

```
while condicion:  
    cuerpo del bucle
```

Cuando llega a un bucle while, Python evalúa la condición y, si es cierta, ejecuta el cuerpo del bucle. Una vez ejecutado el cuerpo del bucle, se repite el proceso (se evalúa de nuevo la condición y, si es cierta, se ejecuta de nuevo el cuerpo del bucle) una y otra vez mientras la condición sea cierta. Únicamente cuando la condición sea falsa, el cuerpo del bucle no se ejecutará y continuará la ejecución del resto del programa.

- While: repetición mientras se ejecuta un bloque mientras se cumpla una condición.



```
i = 1
while i <= 3:
    print(i)
    i += 1
print("Programa terminado")
```

```
1
2
3
Programa terminado
```

```
i = 1
while i <= 50:
    print(i)
    i = 3*i + 1
print("Programa terminado")
```

```
1
4
13
40
Programa terminado
```

```
numero = int(input("Escriba un  
número positivo: "))  
while numero < 0:  
    print("¡Ha escrito un número  
negativo! Inténtelo de nuevo")  
    numero = int(input("Escriba  
un número positivo: "))  
print("Gracias      por      su  
colaboración")
```

```
Escriba un número positivo: -4  
¡Ha escrito un número negativo!  
Inténtelo de nuevo  
Escriba un número positivo: -8  
¡Ha escrito un número negativo!  
Inténtelo de nuevo  
Escriba un número positivo: 9  
Gracias por su colaboración
```

## ► Bucles infinitos

- Si la condición del bucle se cumple siempre, el bucle no terminará nunca de ejecutarse y tendremos lo que se denomina un **bucle infinito**. Aunque a veces es necesario utilizar bucles infinitos en un programa, normalmente se deben a errores que se deben corregir.

```
i = 1
while i <= 10:
    print(i, end=" ")
```

```
1 1 1 1 1 1 1 1 ...
```

```
i = 1
while i > 0:
    print(i, end=" ")
    i += 1
```

```
1 2 3 4 5 6 7 8 9 10 11 ...
```

# Funciones:

Las funciones se pueden crear en cualquier punto de un programa, escribiendo su definición.

- ▶ La primera línea de la definición de una función contiene:
  - ▶ La palabra reservada `def`
  - ▶ El nombre de la función (la guía de estilo de Python recomienda escribir todos los caracteres en minúsculas separando las palabras por guiones bajos)
  - ▶ Paréntesis (que pueden incluir los argumentos de la función, como se explica más adelante)



```
def saludo ():  
    print("Hola")  
    print ("Desde Python")  
    return  
  
print ("Programa prueba no hace nada ")  
saludo()  
  
print ("Termina el programa ")  
|
```



```
print ("Programa prueba no hace nada ")  
saludo()
```

```
print ("Termina el programa ")  
def saludo ():  
    print("Hola")  
    print ("Desde Python")  
    return
```

# Variables locales

- Las variables **locales** sólo existen en la propia función y no son accesibles desde niveles superiores

```
def ejemplo ():  
    a=10  
    print (a)  
    return
```

```
a=5  
ejemplo();  
print (a)  
|
```

```
def ejemplo ():  
    a=10  
  
    return
```

```
ejemplo();  
print (a)
```

# Argumentos y devolución de valores

- Las funciones en Python admiten argumentos en su llamada y permiten devolver valores. Estas posibilidades permiten crear funciones más útiles y fácilmente reutilizables.

```
def suma(x,y):  
    resul=x+y  
    print ("La suma de",a,"+",b," es resultado:",resul)  
    return
```

```
a= 10
```

```
b= 10
```

```
suma(a,b)
```

```
def suma(x,y):  
    resul=x+y  
    return resul
```

```
a= 10
```

```
b= 10
```

```
resultado=suma(a,b)
```

```
print ("La suma de",a,"+",a,"es: ",resultado)
```

```
def suma(x,y):  
    resul=x+y  
    return resul
```

```
print ("Suma de numero")
```

```
a= input("Ingresa un numero: ")  
b= input ("Ingresa otro numero: ")
```

```
resultado=suma(a,b)
```

```
print ("La suma de",a,"+",a,"es: ",resultado)
```

```
def suma(x,y):  
    resul=x+y  
    return resul
```

```
print ("Suma de numero")
```

```
a= int (input("Ingresa un numero: "))  
b= int (input ("Ingresa otro numero: "))
```

```
resultado=suma(a,b)
```

```
print ("La suma de",a,"+",a,"es: ",resultado)
```

# Bibliografía.

- ▶ <https://decode.la/>
- ▶ <https://recursospython.com/>