we design and create professional quality software

Language - Agnostic

Language-agnostic programing

La programación o secuencias de comandos independientes del lenguaje es un paradigma de desarrollo de software en el que se elige un lenguaje en particular debido a su idoneidad para una tarea en particular, y no simplemente por el conjunto de habilidades disponibles dentro de un equipo de desarrollo.

Modern Portfolio Designed

Te quedo? Listo.
(Rodrigo Bueno)

python

PYTHON

Caracteristicas de PYTHON.

- INSTALACION Y ENTORNO
- GESTION DE PAQUETES
- GESTION DE ENTORNOS VIRTUALES
- TIPOS DE DATOS
- CONTROL DE FLUJO
- COLECCIONES
- BUCLES
- FUNCIONES
- RECURSIVIDAD
- FUNCIONES LAMBDA
- PROGRAMACION FUNCIONAL
- EXCEPCIONES
- ARCHIVOS
- MODULOS Y PAQUETES
- TESTING & FRAMEWORK TEST

python

PYTHON

INSTALACION Y ENTORNO

INSTALACION Y ENTORO EN PYTHON.

python

Descarga e instalación

Es primordial para comenzar a desarrollar con Python es tenerlo instalado en nuestra computadora. Veamos cómo conseguirlo.

¿Tengo Python instalado?

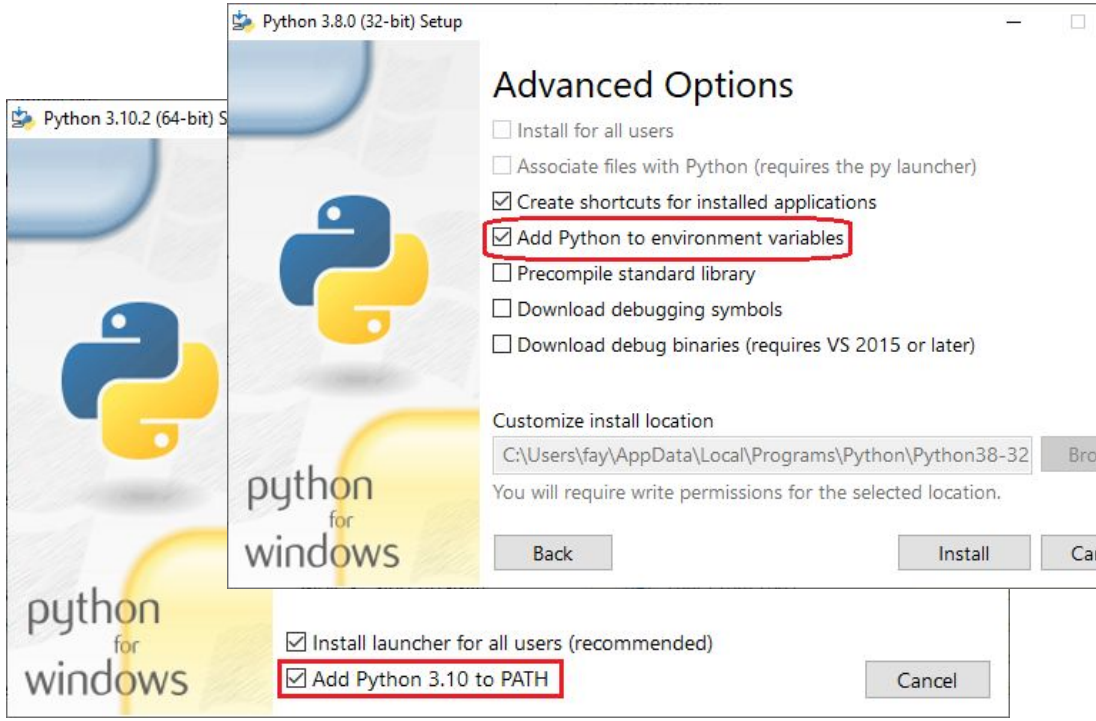
En algunos sistemas operativos, como las distribuciones de Linux y Mac OS, Python estará instalado de fábrica por cuanto es utilizado por herramientas del sistema.
Se puede comprobar abriendo una terminal y escribiendo python --version (en Linux/Mac) o py --version (en Windows). Usaremos la versión de Python 3, . Si tienes una versión de Python 2 es necesario que instales una más nueva (varias instalaciones pueden coexistir en un mismo sistema). Si obtienes un error, entonces Python no está instalado; será mejor pasar al siguiente apartado.

No tengo Python instalado

Para aquellos que no tienen Python instalado de fábrica, vayamos a https://www.python.org/ para descargar la última versión disponible.

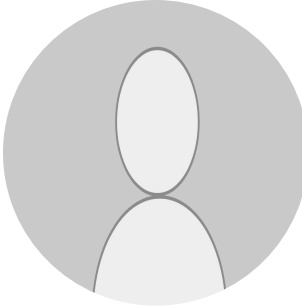
En Windows, obtendremos un feliz instalador que hará todo el trabajo por nosotros. En lo único que debemos prestar atención es en seleccionar el casillero Add Python X.Y.Z to PATH, que nos permitirá abrir Python sin indicar su ruta completa.

Opcionalmente puedes elegir en dónde quieres que se instale Python (presionando en Customize installation). Por practicidad, generalmente se instala —en Windows— en C:\PythonXY\ (considerando que C:\ es la unidad principal).



https://www.python.org/

Primer Script en Python - ¡Hola, mundo!



La instalación de Python incluye un IDLE (un programa en donde escribir el código) llamado IDLE, muy básico, pero que nos será útil en nuestros primeros pasos en el lenguaje. Si ya tienes tu editor de código favorito también puedes usarlo.

Para crear nuestro primer programa vamos a abrir IDLE y seleccionar el menú File > New File para crear un nuevo documento. Luego, escribiremos lo siguiente.

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

Para poder ejecutar este pequeño código primero debemos guardarlo. Para ello, en IDLE vamos a ir al menú File > Save y lo guardaremos en el escritorio como **hola.py**.

Este es un auténtico código de Python. **¡Solo una línea**! En ella llamamos a la función incorporada print() y le pasamos una cadena de caracteres como argumento para que imprima en la pantalla. Se dice que es incorporada ya que es una herramienta que el lenguaje nos pone siempre a disposición en nuestros programas. Existen muchas otras que iremos conociendo en el camino

Ahora bien, recordemos que Python es un lenguaje **interpretado**. Esto quiere decir que no hay un programa compilador que transforme nuestro código fuente (hola.py) y lo convierta en un archivo ejecutable (hola.exe, por ejemplo); más bien, hay un programa llamado **intérprete** al cual le indicamos que queremos ejecutar un archivo determinado. Todos los editores de código pueden hacer esto automáticamente (por ejemplo, en IDLE, presionando F5), no obstante, en este tutorial vamos a hacerlo de la forma manual, esto es, invocando al intérprete desde la terminal. Esto nos dará un panorama más amplio sobre cómo funciona todo en el mundo de Python.

Acceso al IDLE

Acceso al IIDLE en MS Windows

Start PowerShell
wt.bat WSL Linux

```c
#include <stdio.h>
int main(void)
{
    printf("Hello, world!");
}
```

C

```cpp
#include <iostream.h>
int main()
{
    std::cout << "Hello, world! ";
    return 0;
}
```

C++

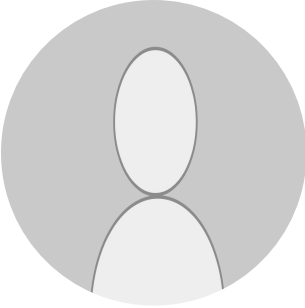
```java
class HelloWorld {
    public static void main(String[]
args) {
        System.out.println("Hello,
World!");
    }
}
```

Java

```
print "Hello, world!"
```

Python

¡Hola, mundo!



Entonces, como decíamos, vamos a abrir la terminal.
Todo sistema operativo tiene algún atajo para esto. En Windows, puedes presionar CTRL + R y escribir cmd, o bien buscar el programa de nombre "Símbolo del sistema". El primer paso es ubicarnos en la ruta en donde hemos guardado nuestro archivo (el escritorio) vía el comando cd. Hecho esto, ejecutamos nuestro script de Python escribiendo python o py seguido del nombre del archivo.

En Linux/Mac:

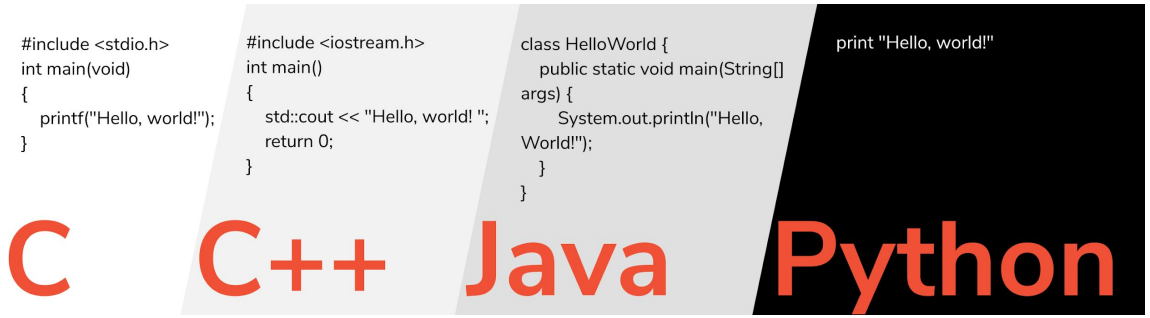
```
> cd Desktop
> python3 hola.py
¡Hola, mundo!
```

En Windows:

```
> cd Desktop
> py hola.py
¡Hola, mundo!
```

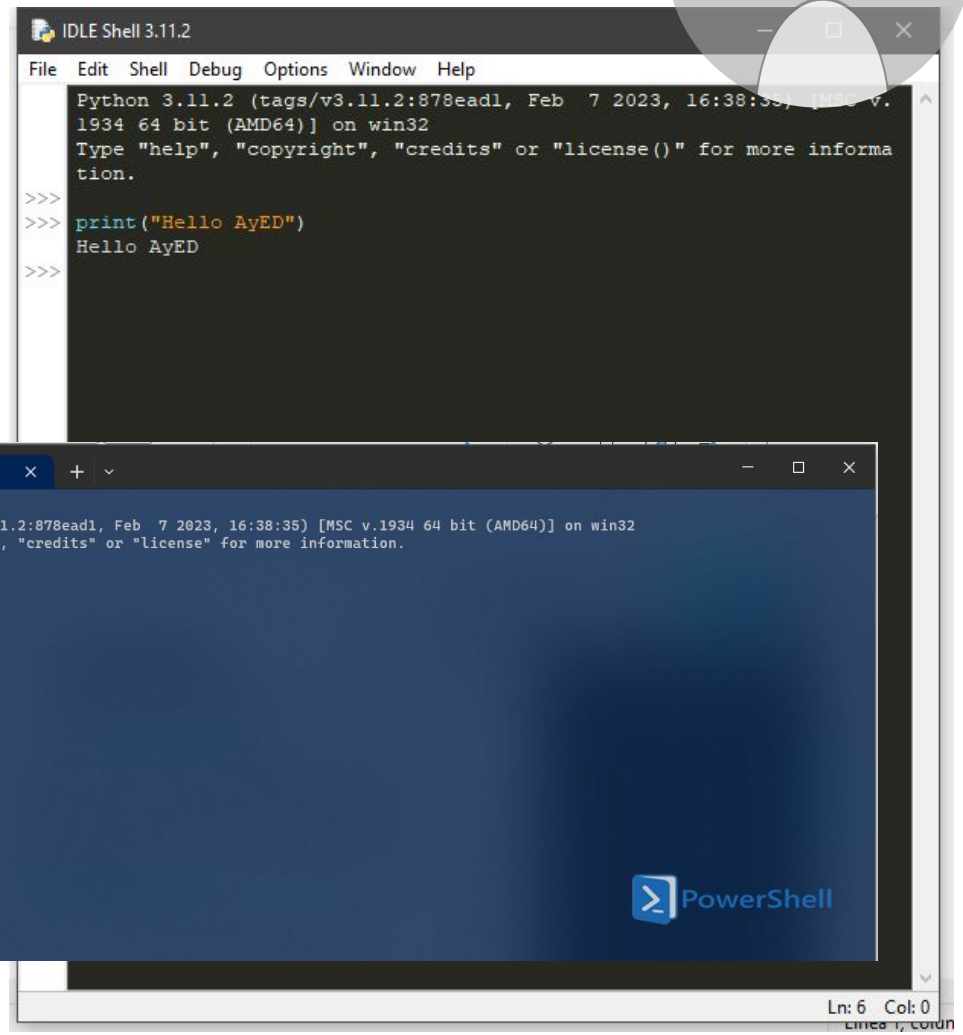
Hemos ejecutado tu primer código de Python.

Recuerda que los programas que escribimos en Python son por defecto aplicaciones de consola. Haciendo doble clic sobre hola.py hará que el intérprete ejecute nuestro archivo, pero una vez impreso el mensaje se cerrará automáticamente (pues es lo que ocurre con todo programa cuando alcanza la última línea de código).



PYTHON

Consola interactiva >>>



La consola interactiva es una herramienta que nos permite escribir sentencias de código de Python al mismo tiempo que se ejecutan. La estaremos usando todo el tiempo, ya que es ideal para probar pequeñas porciones de código. Una vez que la cerramos, todo lo que hemos escrito en ella se pierde.

Para iniciarla vamos a escribir el comando **python** (en Linux/Mac) o **py** (en Windows). Es el mismo que hemos estado utilizando anteriormente, a excepción de que cuando no le indicamos ningún archivo para ejecutar, entiende que queremos iniciar la consola interactiva. Al abrirla deberías ver algo similar a lo siguiente.

```
C:\Users\Administrador>py
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type “
help", "copyright", "credits" or "license" for more information.
>>>
```

Hagamos la prueba de escribir una sentencia que ya conocemos y luego presionemos enter.

```
>>> print("¡Hola, mundo!")
¡Hola, mundo!
```

Como se observa, el código es ejecutado una vez presionada la tecla enter: en este caso, imprime un mensaje en la pantalla. A partir de ahora, todo código que comience con >>> indica que debe escribirse en la consola interactiva.

Acceso al IDLE

Start IDLE MACOS

Acceso a la Consola

python -m idlelib

Start PowerShell

wt.bat WSL Linux

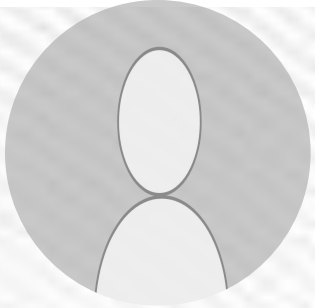
PYTHON

GESTION DE PAQUETES

INTRODUCCION A LA GESTION D EPAQUETES PYTHON.

python

Gestión de dependencias en Python con pip



Un tema que todo desarrollador Python debe aprender tarde o temprano es saber gestionar correctamente las dependencias de sus programas.
Para ello Python nos proporciona la herramienta **pip**

¿En qué consiste la gestión de dependencias?
En Python, tanto programas grandes como scripts se suelen desarrollar apoyándose en librerías y/o frameworks de terceros a los cuales llamamos dependencias. A su vez estas dependencias pueden tener otras dependencias llamadas dependencias transitivas. Esto sucede, por ejemplo, con pandas. Pandas es una librería muy popular pensada para manipular y analizar datos, cuyo desarrollo está basado en NumPy, otra librería muy conocida para realizar cálculos matriciales. Por tanto, al utilizar pandas también utilizamos, aunque de forma indirecta, NumPy.

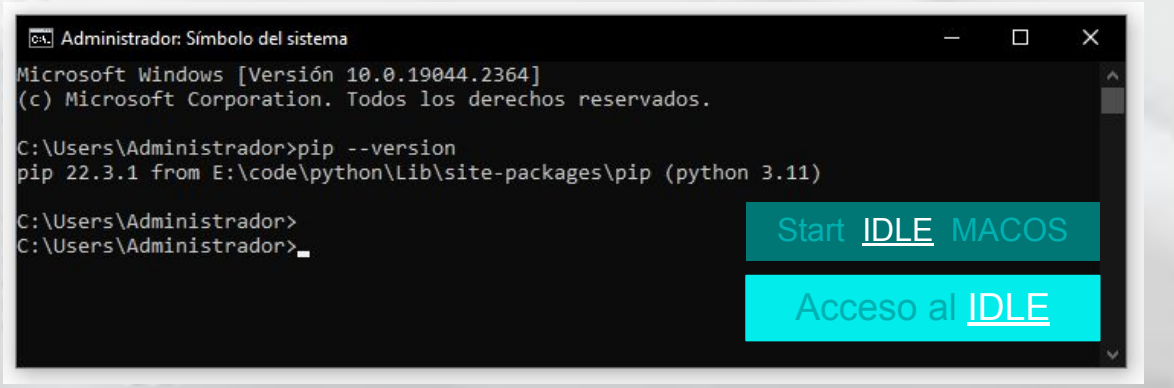
Gestionar todas las dependencias de un programa grande de forma manual puede llegar a ser un proceso tedioso. Por un lado requiere dedicarle tiempo y por otro se trata de un proceso en el cual es fácil realizar fallos. La solución a estos inconvenientes son los llamados sistemas de gestión de paquetes.

Antes de continuar, puntualizar que en Python cuando decimos "paquetes" solemos referirnos a librerías y frameworks de terceros. Algunos ejemplos muy conocidos son, por ejemplo, Django, flask para el desarrollo web, o Requests para realizar peticiones HTTP

.
Python tiene su propio sistema de gestión de paquetes llamado pip. Esta aplicación es en realidad una interfaz de línea de comandos que viene preinstalada con cualquier versión moderna de Python.

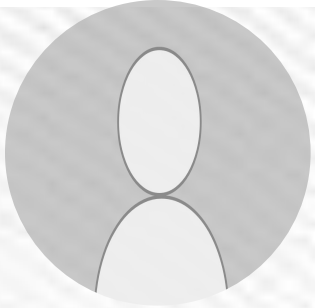
Para saber si lo tenemos instalado es tan sencillo como ejecutar el siguiente comando en nuestro terminal.

$ pip --version
En caso que así sea, nos informará sobre la versión de pip que tenemos instalada. En la siguiente imagen podemos ver que en mi computadora tengo instalada la versión 20.3.4 de pip, la cual es ejecutada por Python 3.9.



Language ***Tips:***
Adicionalmente, también podemos ver un listado de todos los argumentos admitidos por pip mediante el siguiente comando.
$ pip --help

Instalar y actualizar pip



- **Instalación**

Como he comentado en el apartado anterior, pip está incluido por defecto en versiones modernas de Python. En concreto se introdujo a partir de Python 2.7.9 en adelante (en Python 2) y a partir de Python 3.4 en adelante (en Python 3). Si tu versión de Python es anterior a las mencionadas existen dos alternativas para poder instalarlo. La primera, aunque sea un poco obvia hay que mencionarla, y es que actualices a una versión de Python más moderna. Si por el motivo que sea esto no fuera posible, alternativamente puedes añadir pip a tu instalación de Python, pero ten en cuenta que a partir de pip 21.0 se eliminó el soporte para Python 2.

En Windows o macOS, para añadir pip a la instalación de Python, primero tienes que descargar el script de instalación de pip. Luego, ejecutarlo como cualquier otro script.

$ python get-pip.py
En distribuciones de Linux (Debian/Ubuntu) el proceso varía un poco ya que hay que usar su sistema de gestión de paquetes.

$ sudo apt update
$ sudo apt install python3-pip

- **Actualización**

El proceso de actualización de pip varía en función de tu sistema operativo. En Windows y macOS se utiliza directamente el comando pip, mientras que en Linux la actualización se realiza con el sistema de gestión de paquetes. A continuación te detallo como actualizar pip para cada uno de esto tres sistemas operativos.

Windows
C:\>pip install --upgrade pip setuptools
macOS
$ python -m pip install --upgrade pip
Linux
$ sudo apt update && sudo apt upgrade python3-pip

Acceso al IDLE

Start IDLE MACOS

Start PowerShell
wt.bat WSL Linux

python

Repositorios de paquetes de Pythor

- **Repositorios de paquetes de Python**

El repositorio oficial (y más grande) de paquetes de Python es el Python Package Index (PyPI). Cualquier programador puede registrarse gratuitamente en PyPI y subir ahí sus propios paquetes. Una vez que un paquete aparece en PyPI, cualquier persona puede instalarlo localmente mediante pip. Pero como no hay ningún proceso de revisión ni de aseguramiento de la calidad, es recomendado invertir algo de tiempo en revisar el software que estamos adquiriendo.

En PyPI podemos realizar búsquedas de paquetes por nombre y/o filtrar resultados en función de ciertos criterios como el estado de desarrollo, la licencia, etc. Además, cada paquete tiene su propia página en la web de PyPI. Ahí podemos obtener mucha información sobre un paquete en cuestión: la versión, la licencia, el comando para instalarlo, el nombre del autor, las versiones de Python con las que se ha testeado el paquete, y mucho más. Para que te hagas una idea de ello puedes consultar la página de Requests en PyPI.

https://www.pypi.org/

- **Instalar dependencias con pip**

Para instalar con pip cualquier paquete disponible en PyPI, simplemente debes teclear el siguiente comando:

$ pip install nombre
Donde nombre es el nombre del paquete deseado. La herramienta pip no sólo instalará el paquete que le indicamos y su dependencias, sino que además los almacenará en una caché local de caras a futuras instalaciones.

Otro comando útil, complementario al de instalar, es el que nos muestra un listado de los paquetes instalados que es el siguiente:

$ pip list

Una consideración a tener en cuenta es que pip instala por defecto los paquetes en el entorno global de Python. Sin embargo, es recomendado instalar nuestras dependencias en entornos virtuales de Python, ya que de este modo mantenemos las dependencias separadas por proyectos y evitamos posibles conflictos entre versiones de una misma dependencia.

Acceso al IDLE

Repositorios de paquetes de Python

- **Instalación de versiones anteriores**

Por defecto pip instala la última versión disponible de un paquete. Sin embargo, también es posible instalar versiones específicas. Para ello tenemos que especificar la versión deseada como se muestra a continuación, donde instalamos la versión 2.23.0 de la librería Requests:

$ pip install requests==2.23.0
Otra posibilidad es instalar versiones que supongan actualizaciones menores de una versión específica para, por ejemplo, aprovechar la corrección de un bug. El siguiente comando instala la versión 2.18.4 de la librería Requests, que es la más reciente de las versiones 2.18.X.

$ pip install requests~=2.18.0

- **Instalación desde GitHub**

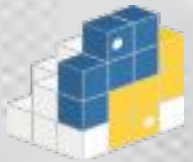
Aunque lo más recomendado es instalar las dependencias directamente desde PyPI, también lo podemos hacer desde sus repositorios en GitHub. Para ello basta con ejecutar un comando tal que así:

$ pip install git+https://github.com/usuario/repositorio.git@rama
De este modo, podemos instalar la versión de la rama máster de Requests con el siguiente comando:

$ pip install git+https://github.com/kennethreitz/requests.git@master
Además, también podemos instalar commits o versiones específicas indicando respectivamente el hash o el número de versión.

Start PowerShell
wt.bat WSL Linux

Acceso al IDLE



Repositorios de paquetes de Python

- **Obtener información de un paquete instalado**

Una vez hemos instalado un paquete en nuestro entorno de Python, podemos obtener información adicional sobre el mismo en nuestro terminal. Para ello tenemos que ejecutar el siguiente comando:

$ pip show nombre

- **Identificar y actualizar dependencias obsoletas**

Con el tiempo, las librerías que tenemos instaladas en nuestro entorno van recibiendo actualizaciones y se quedan obsoletas. Para ver un listado de las librerías que disponen de una nueva versión puedes ejecutar el siguiente comando:

$ pip list –outdated

, Desinstalar dependencias

En algunas ocasiones instalamos una librería simplemente para probarla. Si queremos desinstalarla de nuestro entorno, es tan sencillo como ejecutar el siguiente comando:

$ pip uninstall nombre

Hemos de tener en cuenta que este comando no desinstala las posibles dependencias transitivas. Por eso son tan útiles los entornos virtuales, ya que nos permiten eliminar todas las dependencias de golpe eliminando la carpeta del entorno virtual

- . **Conclusiones**

En este apartado hemos visto a fondo todo lo que concierne a la ***gestión de dependencias en Python con pip***. Desde cómo instalar y actualizar esta herramienta, a cómo utilizarla para instalar, listar, actualizar y eliminar las dependencias que vamos a utilizar para desarrollar nuestros programas o scripts en Python. Otro aspecto que hemos visto es la *importancia de combinar pip con entornos virtuales* para mantener nuestro entorno global de Python limpio, y poder evitar posibles conflictos entre versiones Veamos ahora como gestionar entornos virtuales en Python.

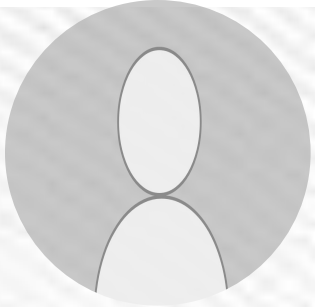
Acceso al IDLE

PYTHON

GESTION DE ENTORNOS VIRTUALES

ADMIONISTRAR ENTORNOS VIRTUALES EN PYTHON.

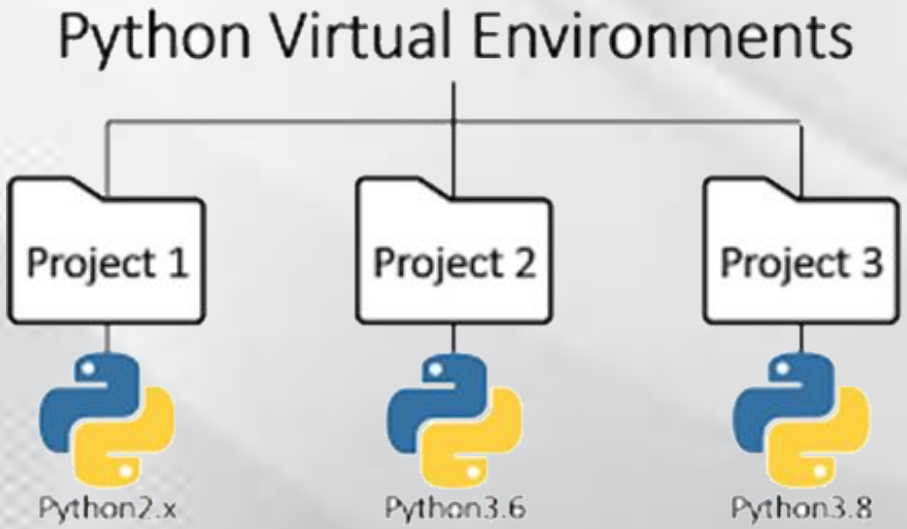
Crear Entornos Virtuales en Python



Un entorno virtual es un espacio donde podemos instalar paquetes específicos para un proyecto. Es decir, permite tener para un proyecto determinado un conjunto de paquetes/librerías aislados de la instalación principal de Python en nuestro sistema. Una razón para usar entornos virtuales es si por ejemplo tenemos muchos proyectos que utilizan una versión específica de una librería, pero queremos probar una versión más reciente de dicha librería sin crear errores en nuestros proyectos existentes.

Para la creación de entornos virtuales se recomienda utilizar el módulo **venv**, el cual viene instalado por defecto con la librería estándar de Python desde la versión 3.3. Por otro lado, la instalación de paquetes se realiza mediante la herramienta pip, la cual viene incluida por defecto a partir de Python 3.4. Como los comandos que vamos a ver en este post utilizan estas dos herramientas, se asume que se trabaja como mínimo con la versión 3.4.

Índice

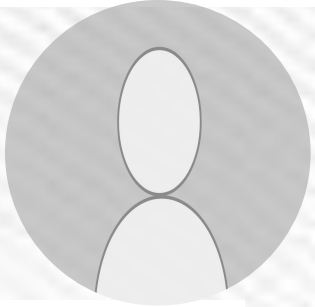


python™
virtualenv

Acceso al IDLE

python

Crear Entornos Virtuales en Python



Lista de comandos | Los siguientes comandos son válidos para el símbolo del sistema (CMD) de Windows, y la aplicación de terminal de macOS y Linux.
Comandos relacionados con la gestión de entornos virtuales

1. **Crear un entorno virtual nuevo**

   En Windows, y asumiendo que Python está incluido dentro de las variables del sistema:
   C:\>python -m venv c:\ruta\al\entorno\virtual

   En macOS y Linux:
   $ python3 -m venv ruta/al/entorno/virtual

   Es recomendado que la carpeta para el entorno virtual sea una subcarpeta del proyecto Python al que esta asociado.

2. **Activar un entorno virtual**

   En Windows:
   C:\>c:\ruta\al\entorno\virtual\scripts\activate.bat

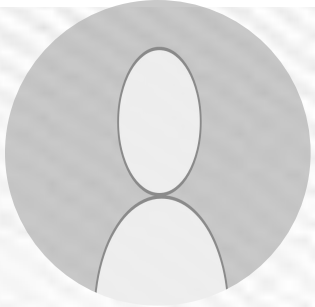
   En macOS y Linux:
   $ source ruta/al/entorno/virtual/bin/actívate

   Sea cual sea nuestro sistema operativo sabremos que el entorno virtual se ha activado porque su nombre aparece entre paréntesis delante del promt.

Acceso al IDLE

python

Crear Entornos Virtuales en Python



Lista de comandos | Los siguientes comandos son válidos para el símbolo del sistema (CMD) de Windows, y la aplicación de terminal de macOS y Linux.
Comandos relacionados con la gestión de entornos virtuales
.

3. Desactivar un entorno virtual

Este comando es idéntico para Windows, macOS y Linux:
$ deactivate

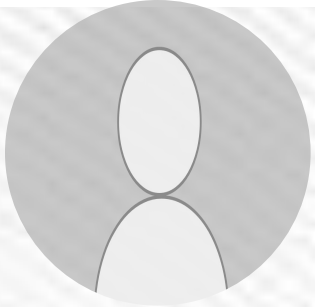
4. Eliminar un entorno virtual

En Windows:
C:\>rmdir c:\ruta\al\entorno\virtual /s

En macOS y Linux:
$ rm -rf ruta/al/entorno/virtual
Eliminar un entorno virtual es tan sencillo como eliminar la carpeta que lo contiene. Por ello, esta operación también se puede realizar desde el correspondiente administrador de archivos.

Acceso al IDLE

Comandos relacionados con la gestión de paquetes



La gestión de los paquetes instalados, sea en el entorno global de Python o en uno virtual, se realiza mediante la herramienta pip.
Esta herramienta obtiene los paquetes que le mandamos instalar del Python Package Index. Para que los siguientes comandos funcionen bajo Windows, se asume que Python está incluido en las variables del sistema.

Instalar una nuevo paquete
$ pip install paquete
Siendo paquete el nombre del paquete/librería a instalar. Este comando instala también automáticamente las dependencias del paquete que deseamos instalar.

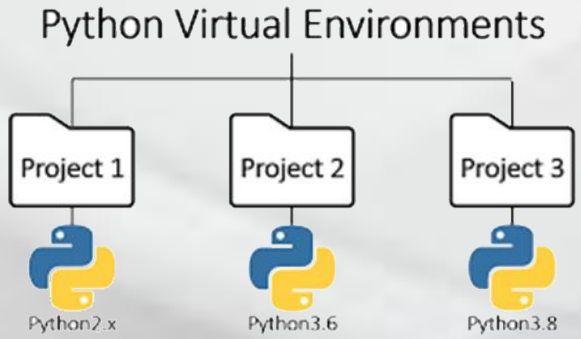
Eliminar un paquete
$ pip uninstall paquete
Siendo paquete el nombre del paquete/librería a desinstalar. Este comando no desinstala cualquier dependencia del paquete que desinstalamos.

Listar los paquetes instalados
$ pip list
Existe otro comando que lista los paquetes en formato del archivo requirements.txt. Este archivo contiene un listado de las versiones de los paquetes necesarios para ejecutar un proyecto en Python.
$ pip freeze

En sistemas Windows podemos copiar la salida de este comando, pegarla en un editor de notas como Atom o Sublime Text (que también podemos usar como nuestro entorno de programación), y guardarla manualmente con el nombre requirements.txt. Sin embargo, si usamos macOS o Linux podemos guardar la salida directamente en un fichero de texto.

$ pip freeze > requirements.txt

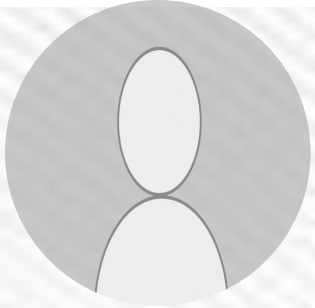
Instalar los paquetes del fichero requirements.txt
$ pip install -r requirements.txt.



Acceso al IDLE

python

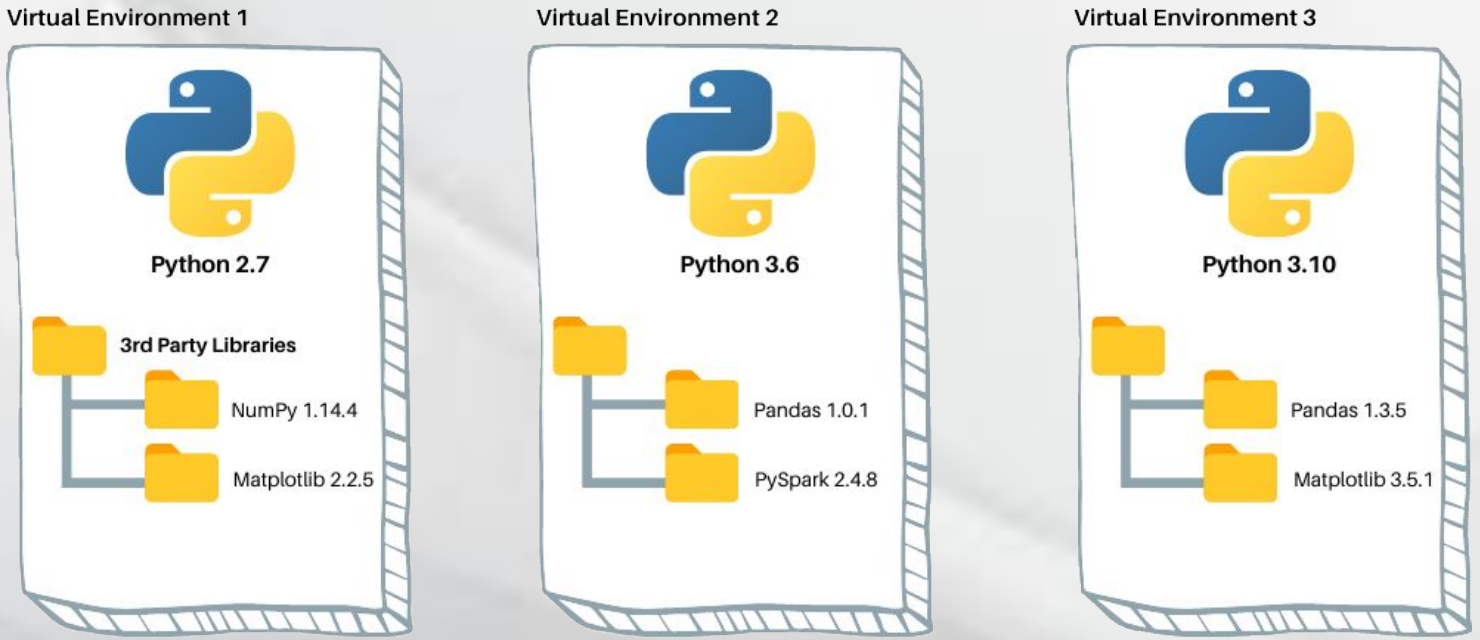
Buenas prácticas con los entornos virtuales



Existen dos consideraciones adicionales a tener en cuenta respecto a los entornos virtuales:

- No se debe añadir manualmente ningún fichero dentro de una carpeta que almacena un entorno virtual. Esta carpeta sólo debe contener los ficheros que se crean por defecto al crear el entorno virtual y los paquetes adicionales que hayamos instalado con el comando pip.
- La carpeta del entorno virtual no debe incluirse a nuestra herramienta de control de versiones. En lugar de ello es recomendado añadir un fichero requirements.txt con el listado de los paquetes necesarios para ejecutar el proyecto.

python virtualenv



Start PowerShell
wt.bat WSL Linux

Acceso al IDLE

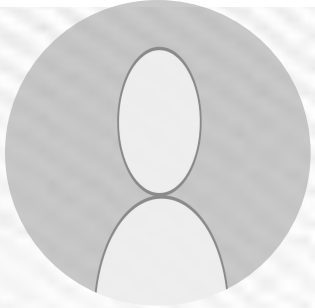
python

PYTHON

TIPOS DE DATOS

INTRODUCCION AL OS TIPOS DE DATOS EN PYTHON.

Tipos de dato



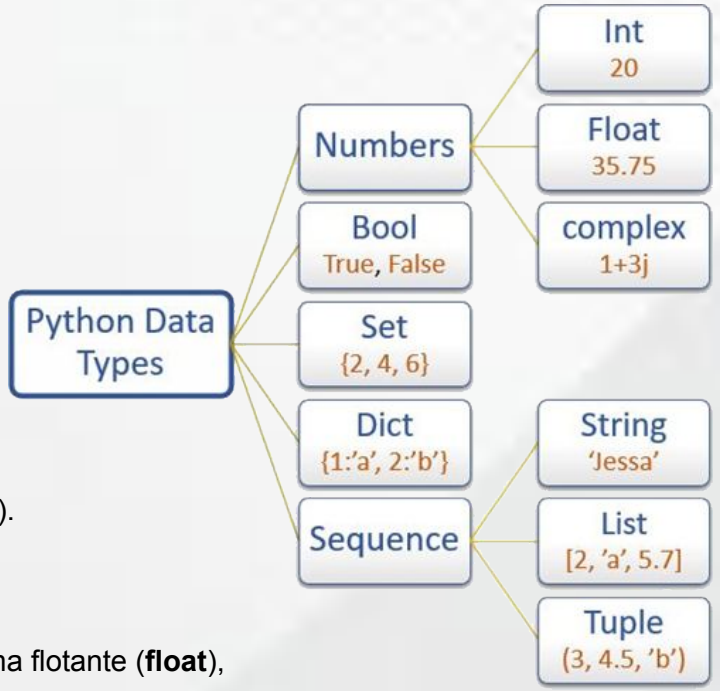
Los lenguajes de programación nos permiten almacenar datos en la memoria del sistema, y ello a través de **variables**. *"Pero una de las diferencias fundamentales entre un lenguaje u otro es el modo en que maneja esas variables"*. De algunos decimos que son **estáticos (C++,Java,C#)**, por cuanto una caja solamente puede almacenar datos de un tipo en particular; mientras que de otros se dice que son **dinámicos (Smaltalk)**, cuando una misma variable puede contener un número al comienzo del programa y luego albergar una cadena de caracteres, por ejemplo.
Según esta clasificación, diremos que ***Python es un lenguaje dinámico***. No obstante, la forma en la que entiende las **"variables"** es un tanto diferente respecto de otros lenguajes. En lugar de concebirlas como "cajas", las entiende como ***nombres, identificadores o referencias***. Por ello, en lugar de "variable" generalmente se emplea el término objeto (aunque bien las utilizaremos de forma indistinta).

Tipos básicos

Hechas las presentaciones, ya es hora de crear nuestro primer objeto. "***Python no distingue la creación de la asignación***".
Un objeto es creado automáticamente cuando le asignamos un valor.
Así, para crear el objeto a con el valor 1 simplemente haremos lo siguiente.

```
>>> a = 1
```

¡Perfecto! Y dado que la creación ocurre de forma implícita en la asignación, tampoco es necesario indicar el tipo de dato que contendrá el objeto pues es *inferido por Python*.



En la consola interactiva, podemos escribir el nombre de un objeto para ver su valor.

```
>>> a
1
```

Para conocer de qué tipo es un objeto, empleamos la función incorporada (recuerda, aquellas que están definidas por Python) type().

```
>>> type(a)
<class 'int'>
```

Como vemos, nuestro objeto es del tipo **int**: esto es, un número entero. Los otros tres tipos de dato básicos son los números de coma flotante (**float**), los booleanos (**bool**) y las cadenas de caracteres (**str**).

```
>>> pi = 3.14
>>> type(pi)
<class 'float'>
>>> prendido = True
>>> type(prendido)
<class 'bool'>
```

```
>>> s = "¡Hola, mundo!"
>>> type(s)
<class 'str'>
```

Nótese que los nombres de las variables se escriben en letras minúsculas y separadas por guiones bajos.

Los valores posibles para un booleano son True y False. Estas dos palabras están reservadas por el lenguaje, lo que quiere decir que no podemos definir objetos con esos nombres.

Language ***Tips:*** Python tiene una peculiaridad y trata sus tipos de datos como mutables o inmutables, lo que significa que si el valor puede cambiar, el objeto se llama mutable, mientras que si el valor no puede cambiar, el objeto se llama inmutable.

Acceso al IDLE

python

Operaciones aritméticas y de comparación

Las operaciones aritméticas son la suma, la resta, la multiplicación y la división. Los operadores utilizados son similares a los de otros lenguajes de programación.

```
>>> a = 5
>>> b = 7
>>> a + b
12
>>> a - b
-2
>>> a * b
35
>>> a / b
0.7142857142857143
```

Todas estas operaciones son expresiones, es decir, sentencias de código que devuelven un resultado. Cuando escribimos una expresión en la consola interactiva, lo que resulta de ello es impreso en la pantalla.

A estas cuatro podemos añadir la división integral y el resto.

```
# División integral: siempre retorna un número entero.
>>> 10 // 3
3

# Resto o módulo: retorna el remanente de la división 16 / 5.
>>> 16 % 5
1
```

En Python, un script es un archivo de texto plano que contiene una serie de instrucciones que se ejecutan en orden secuencial. Estas instrucciones son escritas en lenguaje Python y pueden incluir variables, operadores, estructuras de control de flujo, funciones y clases.
Los scripts en Python se utilizan comúnmente para automatizar tareas repetitivas o para procesar grandes cantidades de datos. Por ejemplo, un script en Python podría leer un archivo de datos, realizar algún tipo de análisis o procesamiento en los datos y luego escribir los resultados en otro archivo.

Ahora bien, en cuanto a comparación de números se refiere, contamos con los operadores > (mayor), >= (mayor o igual), < (menor) y <= (menor o igual). Las operaciones de comparación siempre retornan un booleano.

```
>>> 10 > 5
True
>>> 10 < 5
False
```

```
>>> 6 >= 5
True
>>> 4 <= 2
False
```

Para saber si el valor de dos objetos es igual o distinto, empleamos los operadores == y !=.

```
>>> 5 == 2
False
```

```
>>> "Hola mundo!" != 3.14
True
```

Como habrás observado, no es necesario que los objetos que conforman una comparación sean del mismo tipo de dato.

Arithmetic Operators

a = 5
b = 2

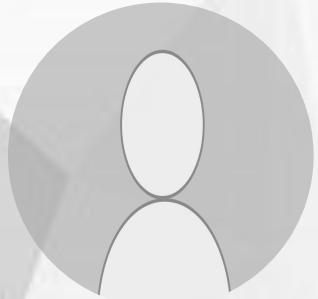
Operator	Meaning	Example	Result
+	Addition Operator. Adds two Values.	a + b	7
-	Subtraction Operator. Subtracts one value from another	a – b	3
*	Multiplication Operator. Multiples values on either side of the operator	a * b	10
/	Division Operator. Divides left operand by the right operand.	a / b	2.5
%	Modulus Operator. Gives reminder of division	a % b	1
**	Exponent Operator. Calculates exponential power value. a ** b gives the value of a to the power of b	a ** b	25
//	Integer division, also called floor division. Performs division and gives only integer quotient.	a // b	2

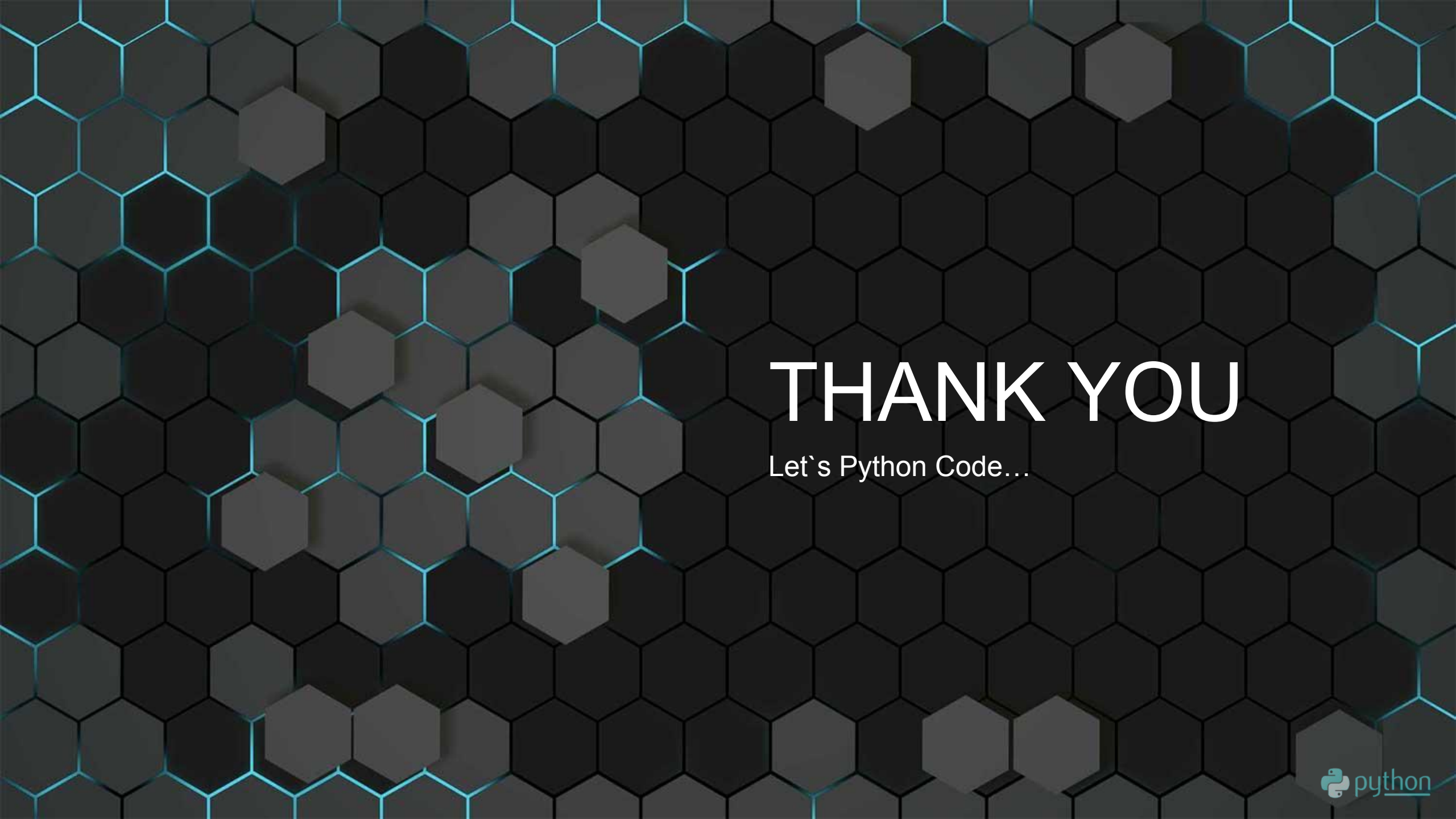
Language - Agnostic

Language-agnostic programing

La programación o secuencias de comandos independientes del lenguaje es un paradigma de desarrollo de software en el que se elige un lenguaje en particular debido a su idoneidad para una tarea en particular, y no simplemente por el conjunto de habilidades disponibles dentro de un equipo de desarrollo.

Modern Portfolio Designed




THANK YOU
Let`s Python Code…
python