



Python de Cero a Experto

Haz algo hoy, conviértete en un experto
que tú futuro te lo agradecerá



Contenido

Introducción

- 1 Breve historia de Python
- 2 ¿Por qué Python?
- 3 ¿Quiénes utilizan Python?
- 4 Versiones y Actualizaciones de Python
- 5 Entorno de Desarrollo
 - 5.1 ¿Qué es un Entorno de Desarrollo Integrado?
 - 5.2 ¿Cuál es el Entorno de Desarrollo Integrado en Python?
 - 5.3 Características de IDLE
- 6 Otros Entornos de Desarrollo
 - 6.1 PyDEV
 - 6.1.1 Características de PyDEV
 - 6.2 PyCharm
 - 6.2.1 Características de PyCharm
 - 6.3 Visual Studio Code
 - 6.3.1 Características de Visual Studio Code
 - 6.4 Sublime Text 3
 - 6.4.1 Características de Sublime Text 3
 - 6.5 Wing Python IDE
 - 6.6 Spyder
 - 6.7 Repl.it
- 7 Librerías
 - 7.1 Librerías estándar de Python
 - 7.1.1 OS

```
Client.getConnectUser(ac  
response) {  
transactions = response.t  
accounts = response.acco  
ser.update({'accessToken':  
{  
$set: {  
userAccount: account,  
userTransactions:  
}  
}, {  
multi: false  
},  
function(err, result)  
console.log(err);  
console.log(result)  
}  
};  
  
res.render('user/account'  
accounts: accounts,  
transactions: transactions  
));  
  
else {  
user.findOne({'accessToken':  
transactions = user.  
accounts = user.user  
res.render('user/account'  
accounts: accounts,  
transactions: transac
```



Contenido

- 7.1.2 GLOB
- 7.1.3 SYS
- 7.1.4 RANDOM
- 7.1.5 DATETIME
- 7.2 Librerías de Terceros
 - 7.2.1 wxPython
 - 7.2.2 Scrapy
 - 7.2.3 Request
 - 7.2.4 Pillow
 - 7.2.5 SQLAlchemy
 - 7.2.6 Numpy
 - 7.2.7 Pygame
 - 7.2.8 NTLK
 - 7.2.9 NOSE
 - 7.2.10 IPython
- 8 Paquetes y Módulos
 - 8.1 ¿Qué es un Módulo?
 - 8.2 Importar Móduo
 - 8.3 ¿Qué es un Paquete?
- 9 Programación Orientado a Objetos
 - 9.1 Variables de instancias o atributos
 - 9.2 Métodos de Instancia
 - 9.3 Variables de Clase
 - 9.4 Herencia
 - 9.4.1 Simple
 - 9.4.2 Múltiple
 - 9.5 Polimorfismo
- 10 Conclusiones

```
req.user = req.user.accounts[0]
Client.getConnectUser(response) {
    transactions = response.transactions
    accounts = response.accounts
    user.update({'accessToken': accessToken})
}
$set: {
    userAccount: account
    userTransactions: transactions
}, {
    multi: false
},
function(err, result) {
    console.log(err);
    console.log(result)
}
);
res.render('user/account', {
    accounts: accounts,
    transactions: transactions
});
else {
    user.findOne({'accessToken': accessToken})
    transactions = user.transactions
    accounts = user.accounts
}
res.render('user/account', {
    accounts: accounts,
    transactions: transactions
});
```

>>>INTRODUCCION

En G-Talent estamos comprometidos con tu aprendizaje es por eso que producimos diversos artículos, e-books, infografías, tutoriales, whitepapers y una infinidad de materiales que te permiten obtener conocimientos prácticos, sencillos y concisos de un tema en particular.

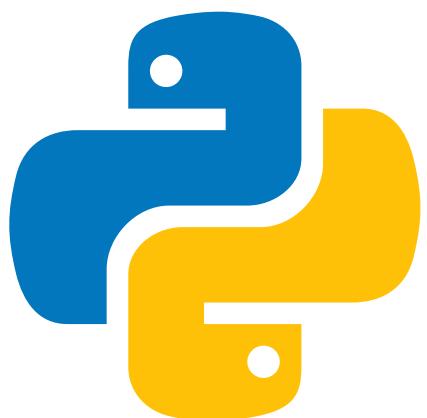
En este apartado vamos a ver las principales herramientas del lenguaje de programación Python.

Python es un lenguaje de programación de código abierto, orientado a objetos, muy simple y fácil de entender. Tiene una sintaxis sencilla que cuenta con una vasta biblioteca de herramientas, que hacen de Python un lenguaje de programación único.

Una de las ventajas principales de aprender Python es la posibilidad de crear un código con gran legibilidad, que ahorra tiempo y recursos, lo que facilita su comprensión e implementación. Así que te hacemos entrega de ésta guía para que te inicies en el mundo de la programación con Python.

Esperamos sea de mucho provecho esta lectura.

G-Talent



>>> BREVE HISTORIA DE PYTHON

Python fue desarrollado por el investigador holandés Guido Van Rossum, quien en aquellas épocas trabajaba en el centro de investigación CWI (Centrum Wiskunde & Informatica) de Ámsterdam.

A finales de los 80, Guido ideó el lenguaje Python y comenzó a implementarlo en diciembre de 1989 , lo bautizó Python en honor a la serie televisiva Monty Python's Flying Circus, de la cual, era seguidor. Esto fue debido al hecho de que visualizó que el principio del diseño del lenguaje fuera divertido de utilizar. En febrero de 1991 publicó la primera versión 0.9.0. La versión 1.0 se publicó en enero de 1994, la versión 2.0 se publicó en octubre de 2000 y la versión 3.0 se publicó en diciembre de 2008.

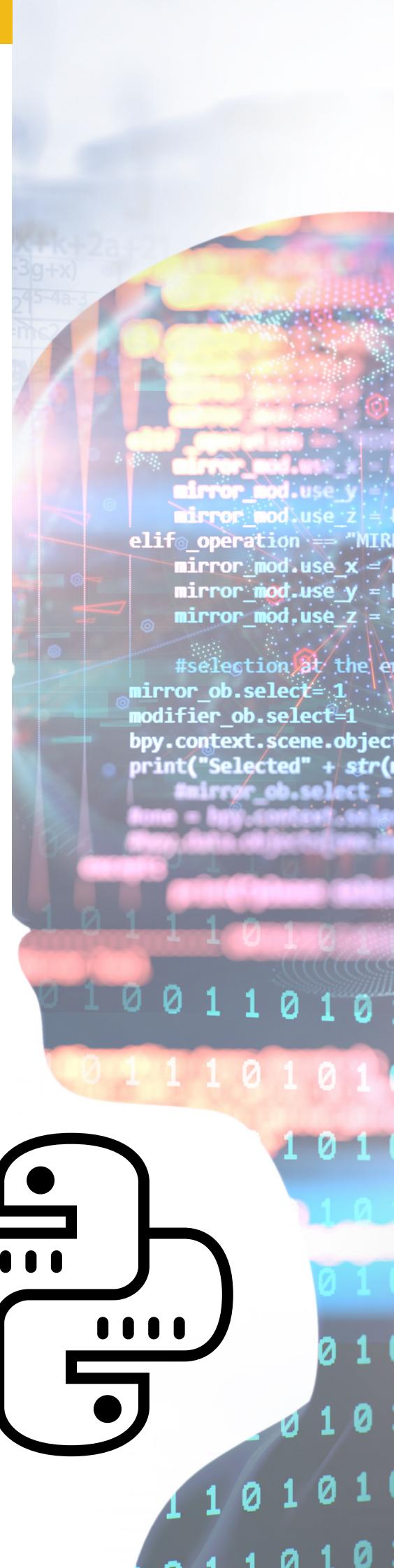
Hasta 2018, el desarrollo de Python estaba dirigido personalmente por Guido van Rossum. A partir del 2019 el desarrollo de Python está dirigido por un consejo de dirección de cinco miembros elegidos entre los desarrolladores de Python y que se renovará anualmente.



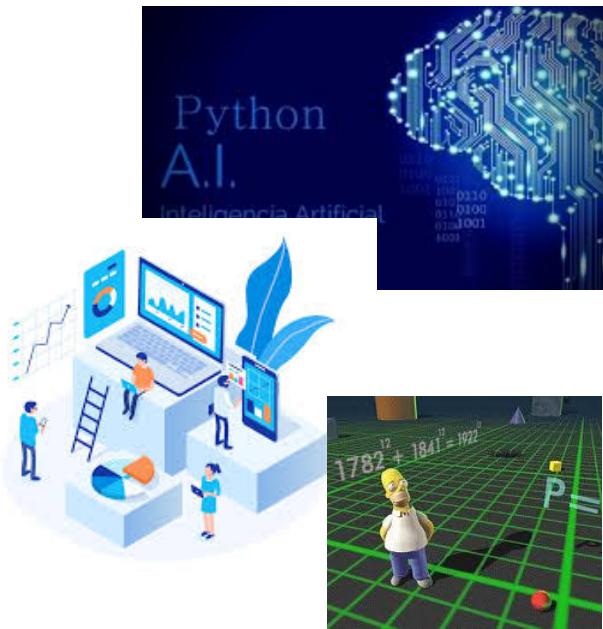
>>> ¿POR QUÉ PYTHON?

Porque es un lenguaje de programación con los siguientes beneficios:

- Interpretado, lo que significa que al ir escribiendo las instrucciones o script, el intérprete la va interpretando y luego la ejecuta.
- Multiplataforma, se puede usar en los diferentes sistemas operativos, entre ellos, Windows, Mac OS, Linux.
- Fácil de aprender con respecto a otros lenguajes de programación, como por ejemplo Lenguaje C, Java, entre otros.
- Es un lenguaje potente, flexible y con una sintaxis clara y concisa.
- Open Source, cualquiera puede contribuir a su desarrollo y divulgación.
- Soporta diferentes paradigmas de programación: Orientado a Objeto, Funcional e Imperativa.



>>> Dónde se utiliza Python?



- Análisis de datos
- Desarrollo WEB
- Desarrollo y operaciones de software (DevOps)
- Pruebas de software
- Educación
- Prototipado de Software
- Programación en red
- Desarrollo aplicaciones de escritorio
- Computación gráfica
- Desarrollo de sistemas integrados
- Desarrollo de juegos
- Desarrollo para móviles
- Inteligencia Artificial
- Machine learning

>>> QUIENES UTILIZAN PYTHON?

En la actualidad Python ha venido tomando auge, cada día son más los entes, instituciones, organizaciones y desarrolladores que se suman al uso del lenguaje de programación, a continuación se nombran algunas redes sociales, organismo que emplean python y software desarrollados bajo python.



Instagram



Netflix



Pinterest



Google



Dropbox



Nasa



Odoo



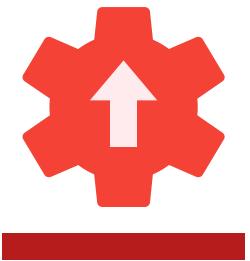
Facebook



Battlefield

>>> VERSIONES Y ACTUALIZACIONES DE PYTHON

Las versiones de Python se identifican por tres números X.Y.Z, en la que:



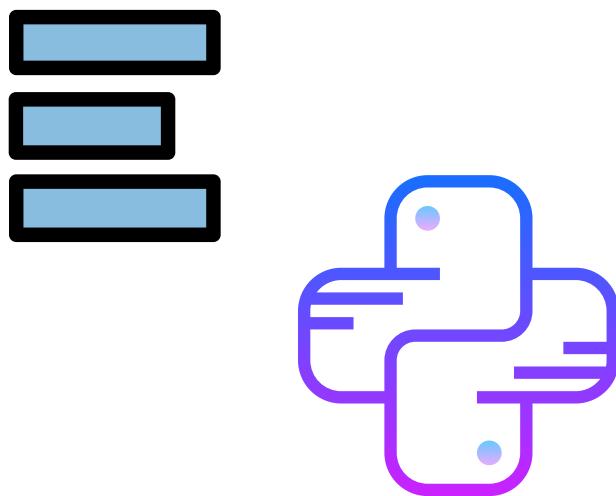
- X corresponde a las grandes versiones de Python (1, 2 y 3)
- Y corresponde a versiones importantes en las que se introducen novedades en el lenguaje pero manteniendo la compatibilidad
- Z corresponde a versiones menores que se publican durante el período de mantenimiento, en las que sólo se corrigen errores durante el primer año y fallos de seguridad en los cuatro restantes.

La última versión recién publicada es la 3.9, se espera para octubre del 2021 sea publicada la versión 3.10, aunque ya hay una versión 3.10.0a1 pre-publicada que correspondería a una versión en mantenimiento e inestable.

>>> ENTORNO DE DESARROLLO

>>> ¿Qué es un entorno de desarrollo integrado?

Es un sistema de software para el diseño de aplicaciones que combina herramientas del desarrollador comunes en una sola interfaz gráfica de usuario (GUI).



>>> ENTORNO DE DESARROLLO

Generalmente, un IDE (Entorno de Desarrollo Integrado) cuenta con las siguientes características:

- Editor de código fuente: editor de texto que ayuda a escribir el código de software con funciones como el resaltado de la sintaxis con indicaciones visuales, el relleno automático específico del lenguaje y la comprobación de errores a medida que se escribe el código.
- Automatización de compilación local: herramientas que automatizan tareas sencillas e iterativas como parte de la creación de una compilación local del software para su uso por parte del desarrollador, como la compilación del código fuente de la computadora en un código binario, el empaquetado del código binario y la ejecución de pruebas automatizadas.
- Depurador: programa que sirve para probar otros programas y mostrar la ubicación de un error en el código original de forma gráfica.

>>> ¿Cuál es el entorno de desarrollo integrado de Python?

Python, provee un editor predeterminado llamado IDLE (Integrated Development and Learning Environment) junto con la venta del Shell de python



>>> Características de IDLE

- Buscar múltiples archivos.
- Tiene un intérprete interactivo con coloración de mensajes de entrada, salida y error.
- Admite sangría inteligente, deshacer, sugerencias de llamadas y autocompletado.
- Le permite buscar y reemplazar dentro de cualquier ventana.

>>> OTROS ENTORNOS DE DESARROLLO

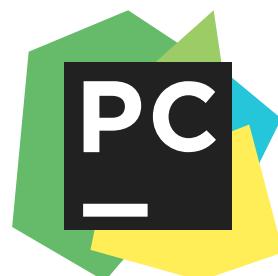


PyDev es un editor de Python de terceros para Eclipse. Este editor puede usarse no solo en Python sino también en el desarrollo de IronPython y Jython.

>>> Características de PyDev

- Tiene atajos de consola interactivos.
- Le permite crear un proyecto Python de Google App Engine (GAE)
- Buscar e ir a la definición
- Importa automáticamente el código para completarlo.
- Puede configurar la integración de Django.

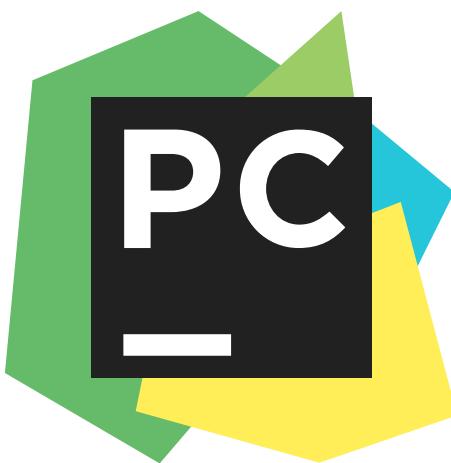
PyCharm, es un entorno de desarrollo creado por la empresa Jet Brains. Viene en dos versiones, la básica que es open source y la podemos descargar del sitio de: JetBrains. Es un IDE multiplataforma utilizado para la programación de Python.



Este editor se puede usar en Windows, macOS y Linux. Este software contiene API que los desarrolladores pueden usar para escribir sus propios complementos de Python para que puedan ampliar las funcionalidades básicas.

>>> Características de PyCharm

- Es un editor de código inteligente de Python compatible con CoffeeScript, JavaScript, CSS y TypeScript.
- Proporciona búsqueda inteligente para saltar a cualquier archivo, símbolo o clase.
- Smart Code Navigation.
- Ofrece una refactorización de código rápida y segura.
- Le permite acceder a PostgreSQL,
- Oracle, MySQL, SQL Server y muchas otras bases de datos desde el IDE.



Visual Studio Code

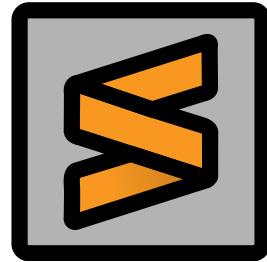
Es un editor de texto que agregando extensiones podemos trabajar con Python con herramientas para programar y depurar programas. Es creado y mantenido por la empresa Microsoft.

Es open source y multiplataforma (Windows, Linux y Mac). Se ha convertido en el editor de texto más utilizado por los programadores profesionales de todo el mundo.

>>> Características de Visual Studio Code

- El editor proporciona terminación de código inteligente basada en la definición de funciones, módulos importados, así como tipos de variables.
- Puedes trabajar con Git y con otros proveedores de SCM
- Le permite depurar el código del editor.
- Proporciona extensiones para agregar nuevos idiomas, depuradores, temas para obtener la ventaja de servicios adicionales.

Sublime Text 3 es un editor de código que admite muchos lenguajes, incluido Python. Tiene soporte básico incorporado para Python.



La personalización de Sublime Text 3 está disponible para crear un entorno de programación Python completo. El editor es compatible con los sistemas operativos OS X, Windows y Linux.

>>> Características de Sublime Text 3

- Le permite resaltar la sintaxis.
- Tiene una implementación de paleta de comandos que acepta la entrada de texto de los usuarios.
- Manejar listas de materiales UTF8 en archivos .gitignore
- Mostrar insignias para carpetas y archivos para indicar el estado de Git
- Los cambios en un archivo están representados por marcadores disponibles en la canaleta.

>>> Otros editores



Es un editor de texto que agregando extensiones podemos trabajar con Python con herramientas para programar y depurar programas. Es creado y mantenido por la empresa Microsoft.

Spyder, es un poderoso entorno científico escrito en Python, para Python, y diseñado por y para científicos, ingenieros y analistas de datos.



Presenta una combinación única de funciones avanzadas de edición, análisis, depuración y creación de perfiles de una herramienta de desarrollo integral con la exploración de datos, ejecución interactiva, inspección profunda y hermosas capacidades de visualización de un paquete científico.

PyScripter es un entorno de desarrollo integrado de Python (IDE) gratuito y de código abierto creado con la ambición de ser competitivo en funcionalidad



con los IDE comerciales basados en Windows disponibles para otros idiomas. Construirse en un lenguaje compilado es bastante más ágil que algunos de los otros IDE de Python y proporciona una amplia combinación de características que lo convierten en un entorno de desarrollo de Python productivo.



Repl.it es una plataforma ideal para programar, tanto si eres un programador experimentado como si estás dando tus primeros pasos en un lenguaje.

Sus puntos fuertes son que no necesitarás instalar absolutamente nada ni pagar para utilizarla. Sólo necesitarás un navegador. La compañía detrás de este proyecto tiene una misión muy clara: que pierdas el menos tiempo posible con configuraciones y empieces a programar cuanto antes.

>>> LIBRERÍAS

>>> ¿Qué es una librería?

Una librería es un conjunto de recursos (algoritmos) prefabricados, que pueden ser utilizados por el programador para realizar determinadas operaciones.

El objetivo es sencillo hacer más fácil y rápido el desarrollo de ciertas funciones dentro de tu aplicación o proyecto.



>>> Librerías estándar de Python

La librería estándar es muy amplia y ofrece una gran variedad de módulos que realizan funciones de todo tipo y vienen instaladas por defecto, a continuación algunas de las librerías:

OS

Permite interactuar con el sistema operativo

```
>>> import os  
>>> os.getcwd()      # retorna el directorio actual  
'C:\\Python39'  
>>> os.chdir('/server/accesslogs')  # cambia el directorio  
actual de trabajo  
>>> os.system('mkdir today')    # ejecuta el comando mkdir  
en el shell  
0
```

Las funciones integradas dir() y help() son útiles como ayudas interactivas para trabajar con módulos grandes como os:

```
>>> import os  
>>> dir(os)  
<returns a list of all module functions>  
>>> help(os)  
<returns an extensive manual page created from the  
module's docstrings>
```

SHUTIL

Permite realizar tareas diarias de administración de archivos y directorios, como puede ser copiar archivos o moverlos

```
>>> import shutil  
>>> shutil.copyfile('data.db', 'archive.db')  
'archive.db'  
>>> shutil.move('/build/executables', 'installdir')  
'installdir'
```

GLOB

Para hacer una búsqueda con comodines en un directorio, provee de una función para hacer listas de archivos

```
>>> import glob  
>>> glob.glob('*.*py')  
'primes.py', 'random.py', 'quote.py']
```

SYS

Proporciona acceso a variables utilizadas o mantenidas por el intérprete y a funciones que interactúan estrechamente con el intérprete. Siempre está disponible.

```
>>> import sys  
>>> print(sys.argv)  
['demo.py', 'one', 'two', 'three']
```

Retorna una lista con todos los argumentos pasados por línea de comandos
El módulo sys también tiene atributos para stdin, stdout, y stderr. Este último es útil para emitir mensajes de alerta y error para que se vean incluso cuando se haya redirigido stdout:

MATH

Permite el acceso a las funciones de la biblioteca C subyacente para la matemática de punto flotante:

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```



RANDOM

provee herramientas para realizar selecciones al azar:

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10)    # sampling without
replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()      # random float
0.17970987693706186
>>> random.randrange(6)     # random integer chosen from
range(6)
4
```

DATETIME

Ofrece clases para gestionar fechas y tiempos tanto de manera simple como compleja. Aunque soporta aritmética sobre fechas y tiempos, el foco de la implementación es en la extracción eficiente de partes para gestionarlas o formatear la salida. El módulo también soporta objetos que son conscientes de la zona horaria.

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%Y. %d %b %Y is a %A on the %d day of
%B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of
December.'
```

```
>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

>>> Librerías de terceros

Python es un lenguaje que viene con un amplio grupo de módulos o librerías en las distribuciones estándares, las cuales a su vez se han ido desarrollando tanto por sí mismas como adecuándose a la evolución del lenguaje.

Aún más grande es la colección de módulos y paquetes desarrollados por la enorme comunidad de programadores de Python en todo el mundo, algunas librerías se presentan a continuación:

wxPython

Es un kit de herramientas de GUI multiplataforma para el lenguaje de programación Python. Permite a los programadores de Python crear programas con una interfaz gráfica de usuario robusta y altamente funcional, de manera simple y sencilla. Se implementa como un conjunto de módulos de extensión de Python que envuelven los componentes GUI de la popular biblioteca multiplataforma wxWidgets , que está escrita en C ++.

Scrapy

Es un marco de aplicación para rastrear sitios web y extraer datos estructurados que se pueden utilizar para una amplia gama de aplicaciones útiles, como minería de datos, procesamiento de información o archivo histórico.

Request:

La librería requests nos permite enviar solicitudes HTTP con Python sin necesidad de tanta labor manual, haciendo que la integración con los servicios web sea mucho más fácil. No es necesario agregar manualmente consultas a las URLs o de convertir información a formularios para realizar una solicitud POST.



Pillow:

Un amistoso fork de PIL (Python Imaging Library). Es mucho más sencillo de utilizar que la propia PIL y se convierte en toda una necesidad para aquellos programadores que trabajen con imágenes.

La biblioteca de imágenes de Python es ideal para aplicaciones de procesamiento por lotes y archivo de imágenes. Puede utilizar la biblioteca para crear miniaturas, convertir entre formatos de archivo, imprimir imágenes, etc.

La versión actual identifica y lee una gran cantidad de formatos. El soporte de escritura se restringe intencionalmente a los formatos de presentación e intercambio más utilizados.

SQLAlchemy:

Una biblioteca muy polémica para gestionar bases de datos. Muchos la aman y muchos la odian. La decisión es cosa tuya.

SQLAlchemy es una librería para Python que facilita el acceso a una base de datos relacional, así como las operaciones a realizar sobre la misma.

Es independiente del motor de base de datos a utilizar, es decir, en principio, es compatible con la mayoría de bases de datos relacionales conocidas: PostgreSQL, MySQL, Oracle, Microsoft SQL Server, Sqlite, ...

Aunque se puede usar SQLAlchemy utilizando consultas en lenguaje SQL nativo, la principal ventaja de trabajar con esta librería se consigue haciendo uso de su ORM. El ORM de SQLAlchemy mapea tablas a clases Python y convierte automáticamente llamadas a funciones dentro de estas clases a sentencias SQL.



NumPy:

De esta librería es muy difícil escapar. Proporciona algunas funcionalidades matemáticas avanzadas para Python.

NumPy es una biblioteca de Python que se utiliza para trabajar con matrices. También tiene funciones para trabajar en el dominio de álgebra lineal, transformada de Fourier y matrices.

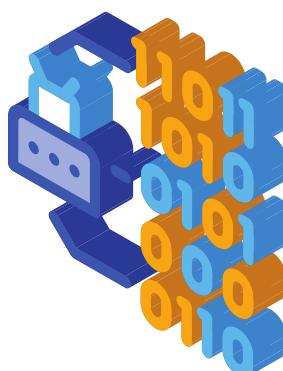
NumPy fue creado en 2005 por Travis Oliphant. Es un proyecto de código abierto y puedes usarlo libremente. En Python tenemos listas que sirven como matrices, pero son lentas de procesar.

Pygame:

¿A qué programador no le gusta echarse unas partiditas a un videojuego? Con la librería Pygame podrás desarrollar juegos en 2D a la antigua usanza.

Pygame(Python Game Library). Librería multiplataforma sobre SDL para Python para la implementación de juegos y aplicaciones de multimedios en 2 dimensiones.

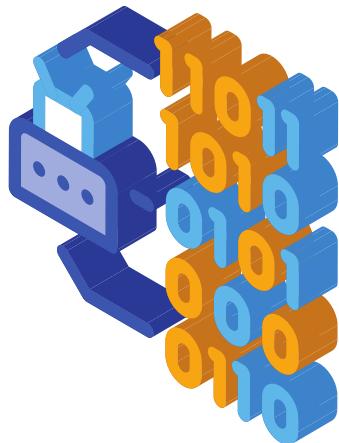
Con sus clases y módulos brinda soporte al desarrollador para importar, tratar y exportar imágenes en varios formatos, IGC y formas básicos, efectos de sonido, reproducción de audio de fondo y CDs, reproducción de video MPEG, tratamiento de eventos de ratón, joystick, teclado, tiempo y otras facilidades que permite rapidez y efectividad para el programador, especialmente si se trata de grupos o empresas pequeños, así como la garantía de soporte para varios sistemas operativos SDL es una librería GNU hecha para el tratamiento de gráficos y contenido multimedia, así como el control de dispositivos como video, teclado, mouse, unidad de CD/DVD, temporizador de la máquina, joystick, etc. Está fundamentalmente orientada a la implementación de juegos sencillos en 2D.



NLTK:

NLTK son las siglas de Natural Language Toolkit. Este kit de herramientas es una de las bibliotecas de PNL más poderosas que contiene paquetes para hacer que las máquinas comprendan el lenguaje humano y respondan con una respuesta adecuada.

NLTK es una librería muy útil si pretendes manipular cadenas, aparte de otras muchas funciones que debes comprobar tú mismo.



NOSE:

Un framework de testing para Python. Es utilizado por millones de desarrolladores de Python. Es totalmente obligatorio si quieres comprobar que todo funcione correctamente en tus proyectos desarrollados con Python.

Nose es una herramienta que nos permitirá ejecutar nuestros tests de una manera sencilla y cómoda

IPython:

No puedo deciros con palabras lo útil que es esta librería para Python. Es como si dijeramos, Python tras una sesión de esteroides. Debes echarle un vistazo, sino estarás perdiendo el tiempo.

IPython proporciona un completo conjunto de herramientas para ayudarlo a aprovechar al máximo el uso de Python de forma interactiva. Sus principales componentes son:

- Un potente shell interactivo de Python.
- Un kernel de Jupyter para trabajar con código Python en cuadernos de Jupyter y otras interfaces interactivas.

>>> PAQUETES Y MÓDULOS

>>> ¿Qué es un Módulo?

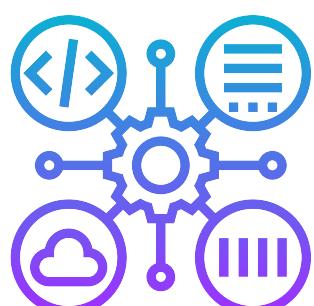
Un módulo es un objeto de Python con atributos con nombres arbitrarios que puede enlazar y hacer referencia. Simplemente, un módulo es no es otra cosa sino un archivo con extensión .py. Un módulo puede definir funciones, clases y variables, también puede incluir código ejecutable.

Un módulo es un fichero conteniendo definiciones y declaraciones de Python. El nombre de archivo es el nombre del módulo con el sufijo .py agregado

Los módulos son entidades que permiten una organización y división lógica de nuestro código.

Los ficheros son su contrapartida física: cada archivo Python almacenado en disco equivale a un módulo.

Un módulo cumple dos roles principales: permitir la reusabilidad de código y mantener un espacio de nombres de variables único.



Importar Módulo

Para usar un módulo tenemos que importarlo a través de la instrucción o sentencia `import <nombre_modulo>`

Veamos un ejemplo:

Creemos un archivo de nombre modulo.py con el siguiente contenido:

```
# modulo.py
def mi_funcion():
    print("una función")

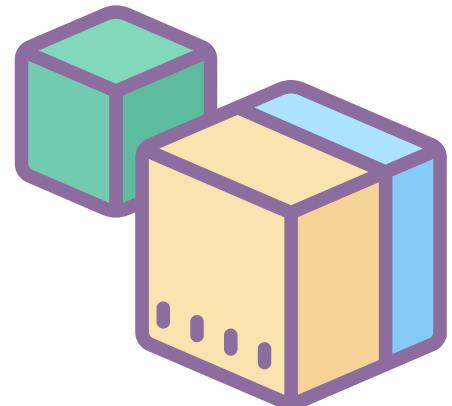
    print("un módulo")
```

Ahora creamos un archivo de nombre programa.py en el mismo directorio en el que guardamos el archivo del módulo, con el siguiente contenido:

```
# programa.py
import modulo
modulo.mi_funcion()
```

Al ejecutar programa.py la salida será:

```
un módulo
una función
```



El import no solo hace que tengamos disponible todo lo definido dentro del módulo, sino que también ejecuta el código del módulo.

La sentencia import también permite importar varios módulos en la misma línea.

```
import os, sys, time
print(time.localtime())
```

Además de la sentencia import, también podemos utilizar from con un comportamiento similar. Ambas difieren en que la segunda nos permitirá utilizar objetos sin necesidad de indicar el módulo al que pertenecen

```
from time import localtime
print(localtime())
```

El operador * puede ser utilizado para importar todos y cada uno de los objetos declarados en un módulo. Así pues, con una única sentencia tendríamos acceso a todos ellos. Por otro lado, si solo necesitamos importar unos cuantos, podemos separarlos utilizando la coma.

```
from time import localtime, asctime, ctime
```

Cuando el intérprete encuentra una sentencia import, este importa el módulo si el mismo está presente en la ruta de búsqueda. Una ruta de búsqueda es una lista de directorios que el intérprete busca antes de importar un módulo.

Secuencia de búsqueda del módulo:

1. El directorio actual.
2. Si el módulo no es encontrado, Python entonces busca en cada directorio en la variable de entorno PYTHONPATH del sistema operativo.
3. Si todas las anteriores fallan, Python busca la ruta predeterminada. En UNIX, la ruta predeterminada normalmente esta /usr/local/lib/python/.

La ruta de búsqueda de módulo es almacenado en el módulo de system sys como la variable sys.path. La variable sys.path contiene el directorio actual, PYTHONPATH, y las predeterminadas dependencia de instalación.

En Python los módulos también son objetos; de tipo module en concreto.

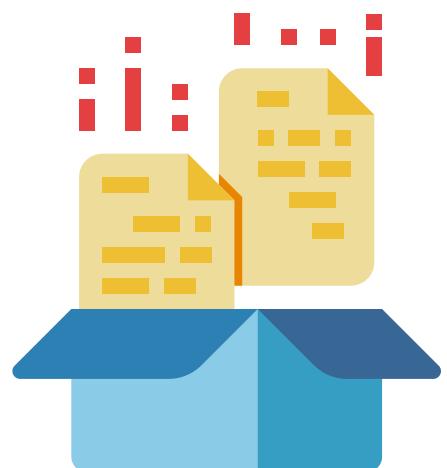
Esto significa que pueden tener atributos y métodos. Uno de sus atributos, __name__, se utiliza a menudo para incluir código ejecutable en un módulo pero que este sólo se ejecute si se llama al módulo como programa, y no al importarlo. Para lograr esto basta saber que cuando se ejecuta el módulo directamente __name__ tiene como valor “__main__”, mientras que cuando se importa, el valor de __name__ es el nombre del módulo:

```
def main():
    pass
if __name__ == '__main__':
    main()
```

>>> ¿Qué es un Paquete?

Un paquete, es un directorio que contiene varios ficheros de código Python que guardan entre sí una relación conceptual basada en su funcionalidad.

La peculiaridad del intérprete es que, automáticamente genera un espacio de nombres de variables a partir de un directorio. Ello también afecta directamente a cada subdirectorio que exista a partir del directorio en cuestión. De esta forma, podemos organizar más cómodamente nuestro código.



Si los módulos sirven para organizar el código, los paquetes sirven para organizar los módulos. Los paquetes son tipos especiales de módulos (ambos son de tipo module) que permiten agrupar módulos relacionados. Mientras los módulos se corresponden a nivel físico con los archivos, los paquetes se representan mediante directorios.

Para hacer que Python trate a un directorio como un paquete es necesario crear un archivo `__init__.py` en dicha carpeta. En este archivo se pueden definir elementos que pertenezcan a dicho paquete, como una constante DRIVER para el paquete bbdd, aunque habitualmente se tratará de un archivo vacío.

Crearemos nuestro primer paquete, vamos a crear un directorio con el nombre `Mi_Paquete`, copiamos en él nuestros scripts `modulo.py` y `programa.py`, luego por último, crearemos un archivo vacío llamado `__init__.py`. Ya tenemos listo nuestro paquete, ahora probaremos su funcionamiento a través de un nuevo archivo (`main.py`), que debe estar fuera del directorio creado, con el siguiente contenido:

```
from Mi_Paquete import modulo  
modulo.mi_funcion()
```

Mi_Paquete/
 __init__.py
 modulo.py
 programa.py
 directorio - paquete nivel alto

Como los modulos, para importar paquetes también se utiliza `import` y `from-import` y el carácter punto (`.`) para separar paquetes, subpaquetes y módulos.

```
import paquete.subpaquete.modulo  
paquete.subpaquete.modulo.función()
```

>>> PROGRAMACIÓN ORIENTADO A OBJETO

La Programación Orientada a Objetos (POO) es un paradigma de programación en el que los conceptos del mundo real se modelan a través de clases y objetos.

La orientación a objetos es ampliamente utilizada por los programadores de Python, especialmente en aquellos proyectos que, por su tamaño, requieren de una buena organización que ayude a su mantenimiento.

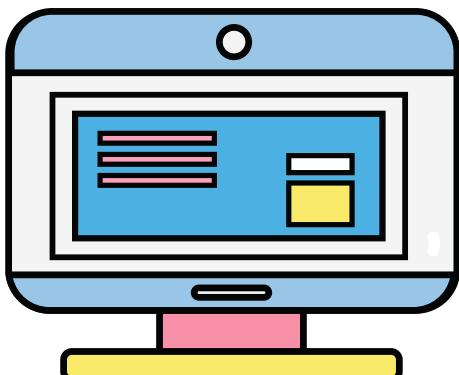
Un objeto es una entidad que agrupa un estado y una funcionalidad relacionados. El estado del objeto se define a través de variables llamadas atributos, mientras que la funcionalidad se modela a través de funciones a las que se les conoce con el nombre de métodos del objeto.



Un objeto es una instancia de una clase. Una clase permite empaquetar atributos o propiedades que son las variables que reciben valores, y métodos que son funciones que contienen sentencias o instrucciones. Viene siendo una plantilla genérica a partir de la cual se instancian los objetos.

En python la clase se define con la palabra reservada `class`, seguido del nombre y los dos puntos(:).

```
class Mi_PrimeraClase:  
    pass
```



Por convención, el nombre de la clase empieza en mayúscula. También suele estar contenida en un fichero del mismo nombre de la clase, pero todo en minúscula.

Para instanciar un Objeto:

```
Obj = Mi_PrimeraClase ()
```

La clase debe contener un método especial con el nombre `__init__` denominado constructor, el cual inicia algunas variables y ejecuta algunos métodos necesarios.

>>> Variables de instancias o atributos

Las variables de instancias deben ir precedidas de la referencia `self` (es un parámetro en el método constructor). Modelemos una clase que contenga unas cuantas variables de instancia:

```
class Moto:  
    def __init__(self, marca, modelo, color):  
        self.marca = marca  
        self.modelo = modelo  
        self.color = color
```

Los atributos de la clase son creados al inicializar o instanciar el objeto

```
Bmw_r1000 = Moto("BMW", "r1000", "rojo")
```

Métodos de instancia

Este tipo de métodos son aquellos que definen las operaciones que pueden realizar los objetos. Los métodos deben tener al menos un parámetro, generalmente se acostumbra escribir la palabra `self` y los demás parámetros requeridos. El parámetro `self`, se emplea para hacer referencias a los atributos de la clase en cada método.

```
def acelerar(self, km):  
    print("acelerando {} km".format(km))
```

La llamada al método quedaría de la siguiente forma:

```
>>> honda_cbr.acelerar(20)  
acelerando 20 km
```

Formalmente, un método de instancia es una función que se define dentro de una clase y cuyo primer argumento es siempre una referencia a la instancia que lo invoca.

```

class articulo():
    def __init__(self,cd,ct,pr):
        self.cod=cd
        self.cant=ct
        self.pre=pr

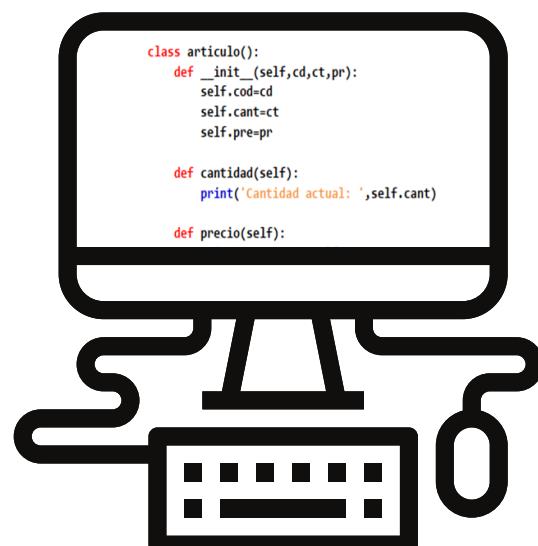
    def cantidad(self):
        print('Cantidad actual: ',self.cant)

    def precio(self):
        print('Precio: ',self.pre)

    def vender(self,x):
        if x<=self.cant:
            self.cant=self.cant-x
        else:
            print('Cantidad insuficiente')

    def comprar(self,x):
        self.cant=self.cant+x

```



Variables de clase

Relacionados con los atributos, también existen en Python las variables de clase. Estas se caracterizan porque no forman parte de una instancia concreta, sino de la clase en general. Esto implica que pueden ser invocados sin necesidad de hacerlo a través de una instancia. Para declararlas bastan con crear una variable justo después de la definición de la clase y fuera de cualquier método

```
class Moto:  
    n_ruedas = 2  
    def __init__(self, marca, modelo, color):  
        self.marca = marca  
        self.modelo = modelo  
        self.color = color
```

Así pues, podemos invocar a la variable de clase directamente:

```
>>> Moto.n_ruedas  
2
```

Además, una instancia también puede hacer uso de la variable de clase:

```
>>> m = Moto()  
>>> m.n_ruedas  
2
```

Herencia

El concepto de herencia es uno de los más importantes en la programación orientada a objetos. En términos generales, se trata de establecer una relación entre dos tipos de clases donde las instancias de una de ellas tengan directamente acceso a los atributos y métodos declarados en la otra.

Para ello, debemos contar con una principal que contendrá las declaraciones e implementaciones. A esta la llamaremos padre, superclase o principal. La otra, será la clase hija o secundaria.

Python implementa la herencia basándose en los espacios de nombres, de tal forma que, cuando una instancia de una clase hija hace uso de un método o atributo, el intérprete busca primero en la definición de la misma y si no encuentra correspondencias accede al espacio de nombres de la clase padre.

Python soporta dos tipos de herencia: la simple y la múltiple.

1. Simple

La herencia simple consiste en que una clase hereda únicamente de otra. La relación de herencia hace posible utilizar, desde la instancia, los atributos de la clase padre.

En Python, al definir una clase, indicaremos entre paréntesis de la clase que hereda.

```
class Padre:  
    def __init__(self):  
        self.x = 8  
        print("Constructor clase padre")  
    def metodo(self):  
        print("Ejecutando método de clase padre")
```

```
class Hija(Padre):  
    def met_hija(self):  
        print("Método clase hija")
```

Instancia de la clase hija

```
>>> h = Hija()  
Constructor clase padre  
>>> h.metodo()
```

El método `__init__()` de clase padre ha sido invocado directamente al invocar la instancia de la clase hija. Esto se debe a que el método constructor es el primero en ser invocado al crear la instancia y como este existe en la clase padre, entonces se ejecuta directamente.

Si creamos el método constructor en la clase hija, este será invocado en lugar de llamar al de padre. Es decir, habremos sobreescrito el constructor original.

```
def __init__(self):  
    print("Constructor hija")  
  
>> z = Hija()  
Constructor hija
```

No solo los métodos son heredables, también lo son los atributos. Así pues, la siguiente sentencia es válida:

```
>>> h.x  
8
```

Varias clases pueden heredar de otra en común, es decir, una clase padre puede tener varias hijas.

```
class Vehiculo:  
    n_ruedas = 2  
    def __init__(self, marca, modelo):  
        self.marca = marca  
        self.modelo = modelo  
    def acelerar(self):  
        pass  
    def frenar(self):  
        pass  
  
class Moto(Vehiculo):  
    pass  
  
class Coche(Vehiculo):  
    pass
```

En este punto podemos crear dos instancias de las dos clases hija y modificar el atributo de clase:

```
>>> c = Coche("Porsche", "944")  
>>> c.n_ruedas = 4  
>>> m = Moto("Honda", "Goldwin")  
>>> m.n_ruedas  
2
```

2. Múltiple

La herencia múltiple es similar en comportamiento a la sencilla, con la diferencia que una clase hija tiene uno o más clases padre. En Python, basta con separar con comas los nombres de las clases en la definición de la misma.

Por ejemplo, una clase genérica sería Persona, otra Personal y la hija sería Cobranzas. De esta forma, una persona que trabajara en una empresa determinada como personal de Cobranzas, podría representarse a través de una clase de la siguiente forma:

```
class Cobranzas(Persona, Personal):
    pass
```

De esta forma, desde la clase Cobranzas, tendríamos acceso a todos los atributos y métodos declarados, tanto en Persona, como en Personal.

POLIMORFISMO

En el ámbito de la orientación a objetos el polimorfismo hace referencia a la habilidad que tienen los objetos de diferentes clases a responder a métodos con el mismo nombre, pero con implementaciones diferentes. Si nos fijamos en el mundo real, esta situación suele darse con frecuencia. Por ejemplo, pensemos en una serpiente y en un pájaro. Obviamente, ambos pueden desplazarse, pero la manera en la que lo hacen es distinta. Al modelar esta situación, definiremos dos clases que contienen el método desplazar, siendo su implementación diferente en ambos casos:

```
class Pajaro:
    def desplazar(self):
        print("Volar")
class Serpiente:
    def desplazar(self):
        print("Reptar")
```

Definiremos una función adicional que se encargue de recibir como argumento la instancia de una de las dos clases y de invocar al método del mismo nombre.

```
>>> def mover(animal):
    animal.desplazar()

>>> p = Pajaro()
>>> s = Serpiente()
>>> p.desplazar()
Volar
>>> s.desplazar()
Reptar
>>> mover(p)
Volar
>>> mover(s)
Reptar
```

>>>CONCLUSIONES

Conclusiones finales

Python es un lenguaje de programación versátil, permite el desarrollo de aplicación en diversos ámbitos como aplicaciones web, aplicaciones de escritorio, aplicaciones móviles, aplicaciones en el área de inteligencia artificial, para cálculos matemáticos, análisis de datos, entre otras. Esa versatilidad lo hace atractivo a los programadores y a las organizaciones que apoyan sus procesos de operatividad y de negocios en aplicaciones desarrolladas bajo este lenguaje.

Además, Python es open source, lo que abarata el costo de licencia para usarlo, presenta una sintaxis bastante sencilla, tiene reglas para escribir el código, lo que le da cierta elegancia y fácil legibilidad. Soporta diferentes paradigmas de programación, uno de ellos es el paradigma de la programación orientada a objeto.

A parte de todas las fortalezas mencionadas anteriormente, es fácil de aprender, debido a su sencillez.

En fin, Python es el lenguaje que muchos quieren aprender, asume el reto y conviértete en un experto en Python.

Curso de Python de Cero a Experto

Puedes adquirir nuestro curso de Python de Cero a Experto y así incursionar en el maravilloso mundo la programación con python.

[Ver curso
Python](#)

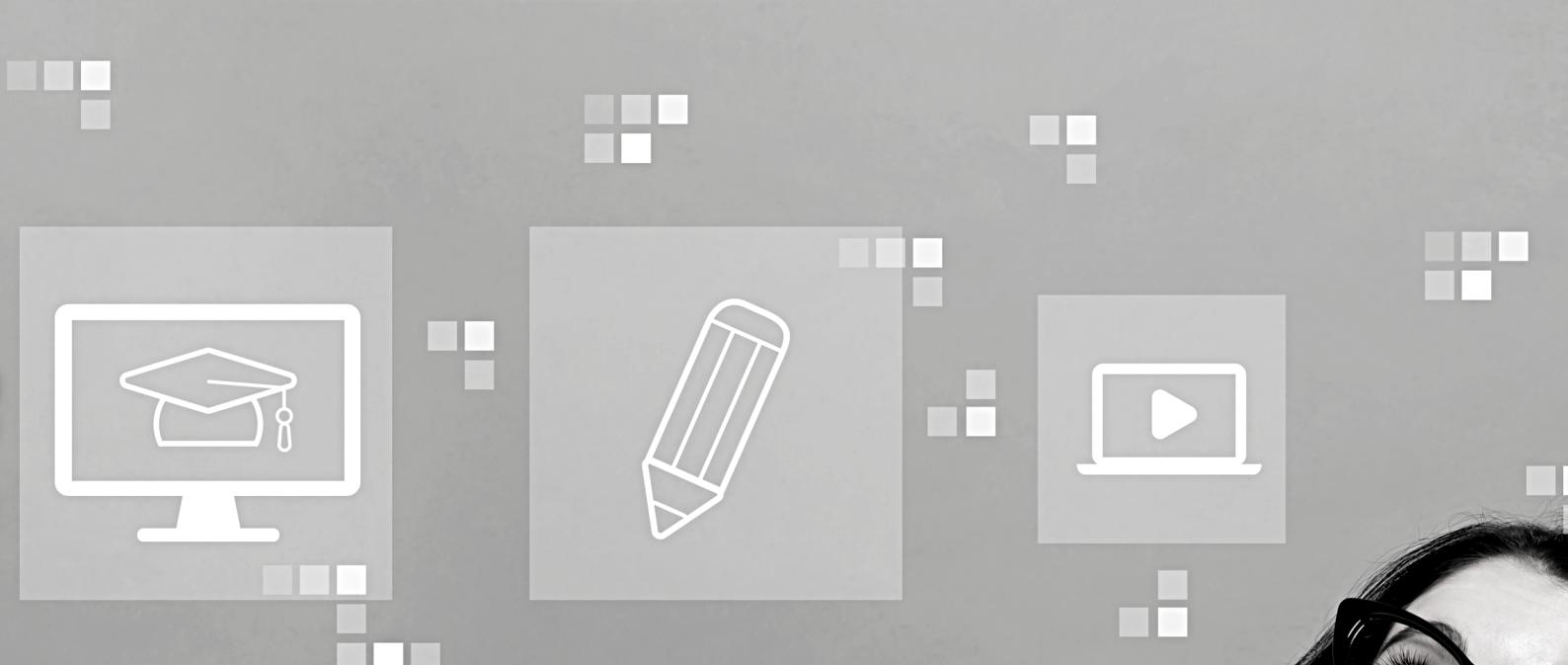


Encontrarás en nuestra Academia cientos de cursos, únete a más de 150 Mil Estudiantes en 15 países de Latinoamerica que hoy estudian en G-Talent.Net

[Visitar
G-Talent.Net](#)



DISPONIBLE EN
 Google Play



Python de cero a experto



Academia de Aprendizaje en línea líder de
Latinoamérica

999 Ponce de Leon, Coral Gables, Miami,
Florida, 33134, United States
info@g-talent.net



G-TALENT.NET

