

Introducción al curso



ÍNDICE

Introducción al curso

1. Introducción al Python
2. Librerías de Python para Machine Learning
3. Machine Learning. Introducción



INTRODUCCIÓN

Los avances tan destacados en las últimas décadas en materia de computación, tanto en *hardware* como en *software*, han propiciado un desarrollo imparable de la Inteligencia Artificial, hasta conseguir que sus desarrollos se encuentren integrados en nuestro día a día. Los avances que observamos en domótica, en asistencia de voz, en los reconocimientos visuales de los móviles, en las recomendaciones que realizan los dispositivos... se incorporan a la existencia de los seres humanos y la facilitan, sin que muchas veces seamos plenamente conscientes de la programación que se ha empleado.

Las técnicas de *Machine Learning* se encuadran dentro del conjunto de desarrollos de Inteligencia Artificial, centrándose en el aprendizaje por parte de las máquinas. Vivimos un cambio de paradigma que posibilita que los dispositivos logren aprender, extrayendo un conocimiento muchas veces escondido, a partir del volumen de datos tan inmensos que podemos manejar, un fenómeno que catalogamos, de manera genérica, como *Big Data*.

A través de esta unidad didáctica, podrás adentrarte en una temática tan apasionante como la del Machine Learning. Comenzarás por realizar una introducción sobre Python, para luego conocer sus librerías y finalizarás centrándote ya en Machine Learning. Seguro que disfrutarás dominando sus distintas técnicas y empleando para ello el lenguaje de programación más popular en nuestros días, Python. ¡Adelante!



OBJETIVOS

- Conocer qué es un lenguaje de programación.
- Identificar las tipologías existentes de lenguajes de programación.
- Estudiar los principales paradigmas de programación.
- Revisar las variables a considerar en la elección de un lenguaje de programación.
- Saber en qué consisten las buenas prácticas en programación.
- Estudiar la evolución histórica del lenguaje Python e identificar sus principales características.
- Saber cómo instalar Python de diferentes maneras.
- Elaborar y comprobar el correcto funcionamiento de un primer mensaje de bienvenida en Python.
- Conocer los principales editores de texto y sistemas IDE para trabajar en Python.
- Diferenciar entre dato, información y conocimiento.
- Conocer qué es una librería en el contexto informático.
- Distinguir entre librería y API.
- Estudiar las ventajas del empleo de librerías.
- Conocer el procedimiento de importación de elementos en Python.
- Identificar los elementos más relevantes en la librería estándar de Python para llevar a cabo desarrollos de Machine Learning.

- Conocer la ubicación de las librerías externas de Python y cómo proceder a su instalación.
- Revisar las principales librerías externas de Python que se pueden emplear para trabajar en el contexto de Machine Learning.
- Revisar el concepto de inteligencia y relacionarlo con el de Inteligencia Artificial.
- Diferenciar distintas tipologías de Inteligencia Artificial.
- Conocer en qué consiste el Machine Learning y su relación con la Inteligencia Artificial.
- Estudiar los condicionantes principales del *Deep Learning*.
- Identificar las tipologías de relaciones entre variables más frecuentes en la configuración de modelos.
- Conocer las diferencias fundamentales entre el modelo de programación tradicional y el enfoque de Machine Learning.
- Analizar las principales tipologías de algoritmos empleados en Machine Learning.

Iconos

En este curso se incluyen recursos didácticos que refuerzan la explicación teórica y te ayudarán a fijar conocimientos y asimilar conceptos. Con estos recursos categorizados, completarás tu proceso de aprendizaje.



¿Sabías que...?



Concepto clave



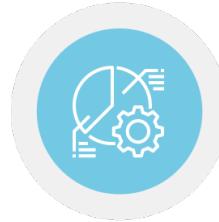
Documento descargable



Vídeo



Consulta



Ejemplo



Caso práctico



Reflexión inicial



Recuerda

1.

Introducción al Python



¿Sabes qué es un lenguaje de programación y cuáles son sus principales aplicaciones? Es muy probable que ya conozcas otros lenguajes de programación, distintos a Python. Lo idóneo es que compares los esquemas de funcionamiento, de manera similar a cuando estudias distintos idiomas. Si no supieras en qué consisten, lo más importante es que valores, desde el primer instante, la idoneidad de generar lo que se conoce como **código limpio**, dado que esto facilitará el desarrollo de las aplicaciones en el ámbito del Machine Learning. Podrás acceder a todos estos conocimientos de manera sencilla y progresiva en esta unidad didáctica, de momento, empezarás por introducirte en **qué consiste un lenguaje de programación**. ¡Adelante!

Los seres humanos empleamos el **lenguaje**, es decir, un conjunto de signos para comunicarnos entre nosotros. Tal como se recoge en la web Significados (s.f.), estos **signos** pueden tener una naturaleza muy diferente y ser, por ejemplo:

- Orales (el habla).
- Corporales (los gestos de la lengua de signos).
- Gráficos (la escritura).



De una manera muy intuitiva, resulta sencillo extrañar este concepto e identificarlo en el sentido de «todo tipo de sistema de señales que permiten comprender un determinado asunto o transmitir un mensaje» (Significados, s.f.).

Bajo este segundo enfoque se puede hablar, por ejemplo, del lenguaje musical o, lo que más nos interesa para nuestro estudio, del **lenguaje de programación** en el contexto de la **comunicación entre las personas y los dispositivos informáticos y móviles** (recuerda en cualquier caso que el lenguaje, en su acepción específica, es un fenómeno que se manifiesta únicamente entre humanos).

¿Cómo definirías, entonces, lenguaje de programación? A continuación, podrás saber en qué consiste.



Un **lenguaje de programación** no es más que el código informático, escrito normalmente por personas programadoras, que permite dar a los ordenadores o *smartphones* una secuencia de instrucciones con el fin de controlar su comportamiento lógico o físico, así como ejecutar determinadas tareas (Santander Universidades, 2021).

Seguro que sabes que existe un elevado número de lenguajes de programación. ¿Por qué pasa esto si, en definitiva, solo hay que proporcionar las instrucciones necesarias a los equipos informáticos para que realicen una serie de actuaciones? Un ordenador o dispositivo similar procesa datos en **lenguaje binario**, que es un sistema de codificación formado únicamente por ceros y unos, y que permite programar cualquier equipo. En principio, bastaría con conocer las reglas de este lenguaje. Sin embargo, trabajar directamente en lenguaje binario resultaría prácticamente imposible para las personas, por su complejidad y laboriosidad.



100
1010
01

A large blue rounded rectangle containing binary code digits (100, 1010, 01) in white.

¿Qué ha sucedido? A lo largo del tiempo, se han ido desarrollando **diferentes lenguajes de programación** que han posibilitado la plasmación de estas instrucciones en programas de manera sencilla y fácilmente legible para el propio individuo que realiza la tarea, así como para otras personas interesadas en su revisión.

El lenguaje de programación empleado ha de ser **entendido por ambas partes**, tanto por la persona que realiza la programación como por el propio equipo. ¿Qué implica esto? Ahora podrás descubrirlo:

- Un esfuerzo de aprendizaje y experimentación por parte de los individuos interesados en adquirir el conocimiento.
- Que, de alguna manera, el código alfanumérico empleado en la elaboración del programa en cuestión se tendrá que transformar en código binario.

El conocimiento de cualquier lenguaje de programación requiere el dominio de las **tres perspectivas** descritas por López (2020):

- **Sintaxis:** es el conjunto de **símbolos** que integran el lenguaje, junto a las reglas que permiten formar sentencias.



- **Semántica:** hace referencia a las **reglas** que permiten transformar dichas sentencias en instrucciones lógicas (a partir de caracteres se forman sentencias y, una vez integradas, estas constituyen instrucciones). La semántica permite comprobar si una determinada sentencia es formalmente válida y si con ella se alcanza el resultado esperado.



- **Pragmática:** implica que el **contexto** afecta de manera relevante al modo en el que se emplea el lenguaje. Una persona que programa de manera pragmática es la que se adapta a los condicionantes del problema y utiliza los lenguajes y las herramientas que mejor se adecúan a la consecución de la resolución.



Tipologías de lenguajes de programación

Ya sabes qué es un lenguaje de programación y qué perspectivas debes dominar para trabajar con él. Ahora es el momento de que identifiques qué **tipos de lenguajes de programación** existen. La clasificación más empleada de lenguajes de programación es la siguiente:

- **Lenguajes de bajo nivel:** se caracterizan, como señala Yáñez (2021), por estar orientados de manera específica hacia la máquina en la que correrá el programa (no se puede transportar entre distintos dispositivos) y por tener un nivel de abstracción hacia el lenguaje humano prácticamente nulo (la visualización de una determinada instrucción no guarda relación con una sentencia o elemento similar empleado en el lenguaje común). Dentro de esta tipología se puede diferenciar, a su vez, otros **dos tipos:**
 - **Lenguaje binario o lenguaje máquina:** como ya sabes, resulta ininteligible para el ser humano.
 - **Lenguaje ensamblador:** posee un cierto nivel de abstracción debido a que cuenta con un conjunto, bastante reducido, de instrucciones que se pueden ejecutar de manera secuencial.



¿Estabas al tanto de que los caracteres que simbolizan cada una de las instrucciones que se emplean en el lenguaje ensamblador reciben el nombre de nemáticos? Dado que el equipo informático o móvil solo entiende las instrucciones en lenguaje máquina se requerirá el empleo de un programa ensamblador, que traduce los nemáticos a código binario.

- **Lenguajes de alto nivel:** su uso se encuentra totalmente generalizado en la actualidad, en contraposición a los de bajo nivel. «Son aquellos que intentan acercarse al lenguaje humano para facilitar al programador su tarea» (Yáñez, 2021). Cada uno de estos lenguajes suele constar de un conjunto, más o menos amplio, de instrucciones (por lo común en inglés), junto a elementos relacionales (operadores aritméticos, lógicos, etc.) y de control del flujo de las operaciones. Se rigen por unos procedimientos de construcción que permiten elaborar, de manera coherente, los correspondientes programas.

Resulta muy interesante la siguiente **subdivisión** de este tipo de lenguajes que se puede establecer en base a la **manera en la que se ejecutan los programas creados** (como comprobarás más adelante esto influye en la instalación de Python):

- **Lenguajes interpretados:** son aquellos cuyo código fuente, es decir, las instrucciones concretas que integran el programa, se va ejecutando **paso a paso**, a medida que el sistema lo va procesando. Se requiere un programa intérprete, que lee la instrucción que corresponda, la traduce y la ejecuta. Esta traducción se llevará a cabo cada vez que el programa es llamado. Python es un lenguaje interpretado, al igual que Java, JavaScript o Perl, entre otros.
- **Lenguajes compilados:** como indica su nombre, requieren una **compilación previa del programa** antes de que se puedan ejecutar. Han de ser traducidos por completo a código binario, de una manera específica. Esta labor la lleva a cabo un programa compilador. Una vez que la compilación se ha llevado a cabo la correspondiente ejecución se puede repetir de manera inmediata, sin que se requiera de un nuevo proceso de traducción. Tanto C como C++ constituyen ejemplos de lenguajes compilados.

Ten en cuenta que los programas escritos en lenguajes interpretados se suelen compilar, una vez que se ha comprobado que no contienen errores, para que funcionen más rápidamente en condiciones de producción real. Esto es muy frecuente en Java y Perl, de ahí que se suelan clasificar en una **categoría híbrida** como lenguajes interpretados/compilados.



En Santander Universidades (2021) se diferencian, a su vez, **tres generaciones** de lenguajes dentro de los que se identifican como de **alto nivel**:

- **Tercera generación:** serían los más antiguos en el tiempo, aunque con modificaciones que facilitan su interpretación. En este bloque se encuadrarían Pascal, C o C++.
- **Cuarta generación:** se caracterizan por su gran proximidad al lenguaje humano. Sería el caso de Java, Visual Basic o Python.
- **Quinta generación:** se englobarían en el ámbito de los denominados lenguajes naturales y se corresponden con los nuevos modelados que en la actualidad se desarrollan en el contexto de la Inteligencia Artificial.

Resulta interesante que visualices la evolución tan relevante que han experimentado los lenguajes de programación comparando, por ejemplo, el código que se requería a mediados de los años 50 del siglo XX para mostrar en pantalla la frase «Hola mundo» frente a cómo se lleva a cabo en la actualidad, empleando un lenguaje de cuarta generación. En la imagen de la izquierda se recoge dicho código en **lenguaje COBOL** (Hola mundo, s.f.), mientras que en la derecha lo tienes en **lenguaje Java** (Lenguajes de programación, s.f.):

COBOL

```

000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.      HOLAMUNDO.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400   DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500   DISPLAY "Hola mundo" LINE 15 POSITION
10.
100600   STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800   EXIT.

```

JAVA

```

class HolaMundo
{
    public static void main(String[] args)
    {
        System.out.println("Hola Mundo");
    }
}

```

Ventajas y desventajas de lenguajes de programación

La proliferación en los últimos años de lenguajes interpretados, como Python, obedece, como es lógico, a sus importantes **ventajas**, tal y como se recoge en la web Lenguajes de programación (s.f.):

VENTAJAS



Multiplataforma



Rendimiento



Modificación del código
en tiempo real

- **Multiplataforma:** las aplicaciones desarrolladas con este tipo de lenguajes se caracterizan por poder funcionar en **diferentes plataformas** y por eso reciben el nombre de aplicaciones multiplataforma. Dado que el programa intérprete suele estar implementado en distintos sistemas operativos no existe la necesidad de adaptar el código a cada posible sistema de destino.
- **Rendimiento:** el rendimiento de dichas aplicaciones suele **aumentar**, en especial en el ámbito web. El programa se suele ejecutar en el navegador de la persona que accede al portal, por lo que se reducen los requerimientos del servidor en el que se encuentra instalada la aplicación.
- **Modificación del código en tiempo real:** como exponen otras personas especialistas en la materia, el empleo de lenguajes interpretados posibilita modificar el código en tiempo real, lo que permite introducir pequeños cambios en los programas y ver los **resultados de manera prácticamente inmediata**, sin que se requiera la compilación completa de la aplicación de la que se trate cada vez que se incorpore una modificación.

Las principales **desventajas** de estos lenguajes se encuentran relacionadas con los siguientes hechos:

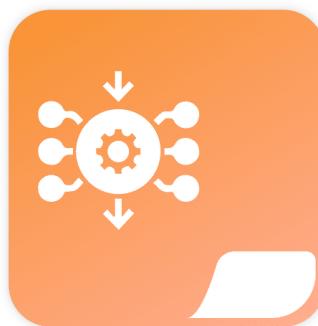
DESVENTAJAS



Pérdida de velocidad de ejecución



Fallos de funcionamiento



Máquina virtual

- **Pérdida de velocidad de ejecución:** al menos, en comparación con los lenguajes compilados. Este aspecto puede resultar crítico o no, en función de la aplicación de la que se trate, por lo que la persona programadora debe **analizar su importancia** y adoptar la decisión correspondiente sobre la tipología de lenguaje a emplear.
- **Fallos de funcionamiento:** los lenguajes interpretados también se suelen caracterizar por una mayor **dificultad en la identificación de los fallos** de funcionamiento o bugs. Como comprobarás a través de tu experiencia, el programa colapsará en cuanto se produzca un error, sin que el programa intérprete continúe traduciendo otras instrucciones.
- **Máquina virtual:** pueden darse problemas asociados a la necesidad de que el sistema tenga cargada una máquina virtual que actúe como **intérprete de las instrucciones** del programa.

Paradigmas de programación

Ya conoces los principales tipos de lenguajes de programación, así como sus ventajas y desventajas, pero todavía necesitas profundizar mucho más en ellos. A continuación, te centrarás en un aspecto específico que también caracteriza a los lenguajes de programación. Se trata del **paradigma** en el que se basan o que adoptan.

Como indica Martínez (2020), «un paradigma de programación es una manera o estilo de programación de software». Como analizarás seguidamente, existen **diferentes enfoques** a la hora de diseñar el funcionamiento de un lenguaje de programación, de modo que se facilite la tarea de las personas que realizan la programación de las aplicaciones.

En cierto modo, un paradigma de programación se identificaría con el **modelo genérico** que resulta recomendable emplear para **construir un programa**, teniendo en cuenta el diseño del lenguaje en cuestión.



Martínez (2020) identifica los siguientes **paradigmas** que constituyen, como acabas de comprobar, otro elemento de **clasificación de lenguajes de programación**. A continuación, conocerás estos paradigmas:

Paradigma **imperativo o procedural**

Los programas escritos bajo este enfoque están integrados por un conjunto de instrucciones planteadas para su ejecución secuencial «como si el programador diera órdenes concretas», de ahí el término imperativo, es decir, algo que se **impone**. En la imagen que tenías anteriormente, y que puedes volver a analizar ahora, en lenguaje COBOL puedes comprobar cómo hay una parte del código en la que se ordena al sistema que “comience” (BEGIN), que muestre en pantalla un mensaje en una determinada posición y que, finalmente, concluya la operativa (STOP RUN).

Bajo el marco del paradigma imperativo se diferencian, a su vez, **tres enfoques** particulares:

- **Paradigma estructurado:** que se caracteriza por el hecho de que el flujo de control del programa se gestiona a través de bucles, bloques condicionales (que se ejecutan solo si se cumple alguna condición) y subrutinas.
- **Paradigma procedimental:** que implica que la aplicación se articula en torno a un conjunto no muy amplio de instrucciones, que se repiten de manera frecuente. Para dotar de mayor agilidad al programa lo habitual consiste en englobar estas instrucciones en un determinado procedimiento, al que se llama cada vez que resulta necesario.
- **Paradigma modular:** que consiste en una evolución del planteamiento anterior. En este caso el programa se divide en una serie de módulos o pequeños programas a los que se va llamando a medida que se ejecuta la aplicación principal. Este enfoque facilita, además, la legibilidad del código resultante.

Los lenguajes de programación más conocidos que emplean el enfoque imperativo son COBOL, FORTRAN y C.

```
000100 IDENTIFICATION DIVISION.  
000200 PROGRAM-ID. HOLAMUNDO.  
000300  
000400+  
000500 ENVIRONMENT DIVISION.  
000600 CONFIGURATION SECTION.  
000700 SOURCE-COMPUTER, RM-COBOL.  
000800 OBJECT-COMPUTER, RM-COBOL.  
000900  
001000 DATA DIVISION.  
001100 FILE SECTION.  
001200 PROCEDURE DIVISION.  
100100  
100200 MAIN-LOGIC SECTION.  
100300 BEGIN.  
100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS.  
100500 DISPLAY "Hola mundo" LINE 15 POSITION 10.  
100600 STOP RUN.  
100700 MAIN-LOGIC-EXIT.  
100800 EXIT.
```

Paradigma **declarativo**

Las sentencias del programa se emplean para describir los **condicionantes del problema** que se desea resolver, pero no para indicar (como sucedía en el caso anterior) cómo obtener la solución. El resultado final se alcanza mediante la aplicación de **principios internos de razonamiento lógico**.

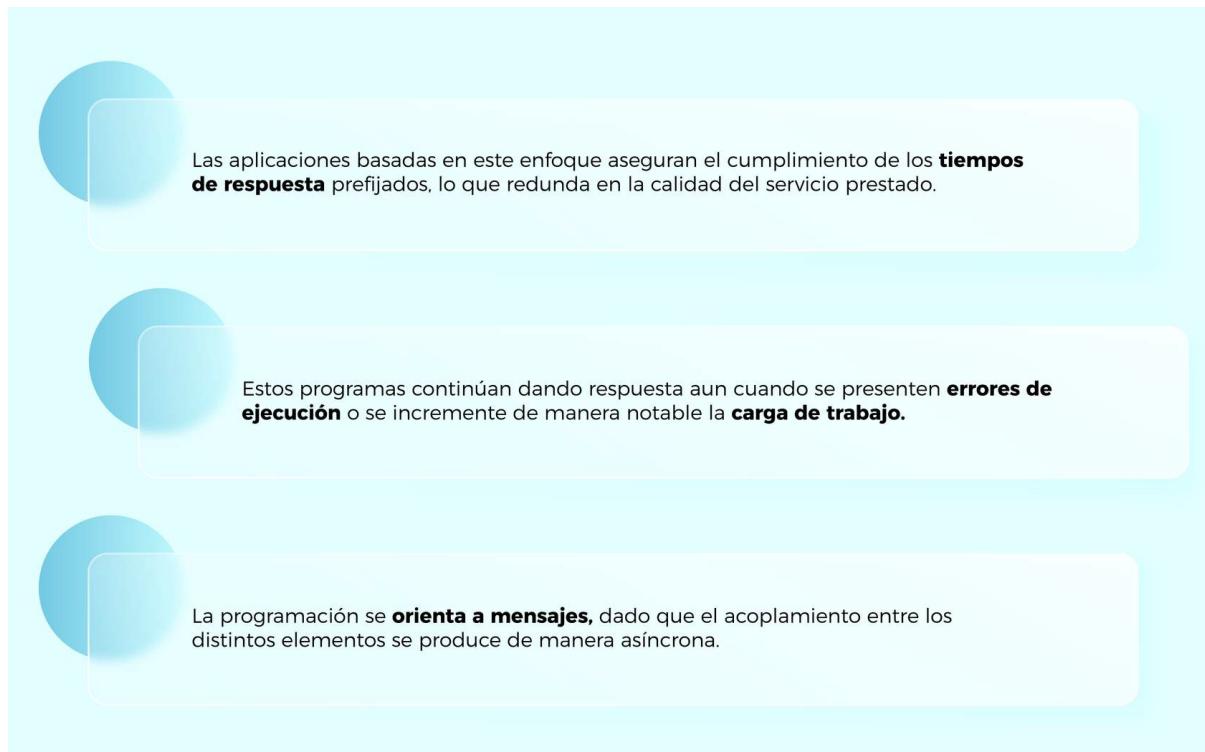
Dentro de este esquema se encuentran las dos siguientes **divisiones**:

- **Paradigma lógico:** basado en la generación de conocimiento mediante axiomas declarativos y en la resolución de una cuestión a través de mecanismos de inferencia. El lenguaje de programación representativo de este enfoque es Prolog.
- **Paradigma funcional:** que implica el empleo de funciones que pueden adoptar distintas formulaciones (algo que resulta especialmente útil en el tratamiento de tareas con un alto grado de abstracción y, por tanto, muy complejas). La aplicación de este paradigma ha experimentado un crecimiento muy destacado en los últimos años, sobre todo en el desarrollo de aplicaciones de naturaleza técnica, aunque se recomienda su empleo únicamente para tareas específicas, que no requieran, por ejemplo, la conexión a servidores o a bases de datos externas. Lisp, ML o Scala son lenguajes de programación que se basan en este paradigma, aunque Perl, Ruby o incluso Java también permiten utilizar dicho enfoque.

Paradigma **reactivo**

Se caracteriza por el hecho de que la aplicación es sensible a la **propagación de los datos generados**, de modo que, en función de estos, puede realizar cambios en el propio programa (denominados eventos). Esto permite, de manera genérica, liberar recursos que no se están empleando y **aumentar la eficiencia** de las aplicaciones activas.

Pulido (2019) relaciona los principales **elementos distintivos** de dicho paradigma. Presta atención a la siguiente imagen para identificarlos:



Java permite el empleo del paradigma reactivo mediante la implementación de una de estas dos librerías: RxJava y Project Reactor. Por su parte, JavaScript también incorpora dicha visión mediante un framework específico, como puede ser React+redux-saga o NgRx.

Paradigma **de programación orientada a objetos**

Paradigma de programación orientada a objetos: este planteamiento se basa en la construcción de modelos que tratan de simular las características y el funcionamiento de objetos existentes en la realidad (o que se pueden diseñar de este modo). El paradigma, con un amplísimo campo de aplicación en la actualidad, facilita también la legibilidad del código, así como la detección de errores y la incorporación de mejoras.

Los principales lenguajes de programación que adoptan el paradigma de orientación a objetos son Java, JavaScript, PHP y Python. Fíjate en que algunos de estos lenguajes también pueden emplear otros enfoques, de ahí que reciban la denominación de lenguajes **multiparadigma**. Lo habitual es que las aplicaciones escritas con estos lenguajes se construyan a través de la combinación de estructuras basadas en modelos diferentes, lo que permite aprovechar las correspondientes funcionalidades de cada paradigma.



Cuatro principios de la programación orientada a objetos



Aunque en este curso emplearás con un enfoque muy específico el paradigma de programación orientada a objetos (que es uno de los que puede adoptar Python), es recomendable el siguiente vídeo como **introducción a las características** de un enfoque muy utilizado en la actualidad. Lo importante es que captes el **enfoque genérico de este paradigma**.

Variables a considerar en la elección de un lenguaje de programación

¿Alguna vez te has preguntado cuál es el lenguaje de programación sobre el que deberías centrar tu esfuerzo? Esta cuestión y su respuesta están influenciadas por la tipología y el objetivo de los programas a desarrollar.



En el **contexto profesional**, como resulta lógico, la elección del lenguaje de programación que se va a emplear para, por ejemplo, desarrollar una aplicación dependerá del **campo de actividad** de que se trate, por ejemplo, los requisitos a cumplir para la correcta gestión de inventarios son muy diferentes a los que deberá verificar una aplicación orientada al cálculo de estructuras de edificios, aun cuando el lenguaje finalmente empleado por las personas que realizan la programación puede ser el mismo.

El tratamiento de esta compleja y apasionante problemática se reduce en nuestro curso, dado que nuestro objetivo es el análisis de técnicas de **Machine Learning**, que es un campo en el que **Python** posee un amplio potencial. Aun así, es interesante realizar una breve revisión de cuáles son los aspectos de un lenguaje de programación que las personas expertas valoran en mayor medida y que propician su **popularidad** durante, al menos, un cierto período de tiempo.

Siguiendo las recomendaciones de Laura M. (2021) de la web de BitDegree y de la web Chakray (s.f.), se identifican los siguientes **factores** que pueden resultar decisivos a la hora de seleccionar un determinado lenguaje de programación para el desarrollo de una aplicación:

| | |
|--------------|----------------------|
| Versatilidad | Multiplataforma |
| Singularidad | Nivel de abstracción |
| Simplicidad | Eficiencia |
| Naturalidad | Compacidad |

- **Versatilidad:** la capacidad de empleo para la resolución de un conjunto amplio de problemáticas. Este aspecto animará al estudio y especialización por parte de las personas interesadas en actividades profesionales en el campo de la programación.
- **Singularidad:** aunque pueda parecer contradictorio con lo indicado anteriormente, hay lenguajes de programación muy demandados precisamente por resultar muy específicos para el tratamiento de ciertas temáticas. Es lo que sucede, por ejemplo, con SQL en el campo del tratamiento efectivo de bases de datos.
- **Simplicidad:** que redundará en que resulte sencillo de aprender y, por tanto, fácil de comprender, de depurar, de incorporar modificaciones, etc. La web de BitDegree destaca, en este sentido, la rápida expansión del lenguaje Swift en el mundo Apple, debido a la facilidad de uso frente a Objective-C, el estándar empleado hasta no hace mucho en el entorno iOS.
- **Naturalidad:** que se pueda aplicar de manera eficiente, sin necesidad de incorporar mecanismos forzados de ajuste, en el campo de actividad sobre el que se esté trabajando.
- **Multiplataforma:** estar soportado por la mayoría de las plataformas (en especial iOS, Google o Android). Este es un factor crucial de cara a contar con garantías de adopción por una amplia mayoría de las personas expertas en programación.
- **Nivel de abstracción:** el suficiente nivel de abstracción para facilitar el empleo de bloques de programación complejos.
- **Eficiencia:** en el sentido de que un potente lenguaje de programación no debe ocupar demasiado espacio en memoria ni ha de requerir elevados tiempos de ejecución.
- **Compacidad:** capacidad de que las operaciones se puedan expresar de manera concisa, sin excesivos detalles (recuerda la comparación de la extensión del código COBOL y la del código Java, que analizaste anteriormente, para obtener el mismo resultado).



¿Te has quedado con ganas de seguir indagando sobre los lenguajes idóneos para aprender a programar? En ese caso, puedes hacer clic en este enlace de Computer Hoy [«8 lenguajes de programación para aprender a programar desde cero»](#) y en este otro de Xataka [«17 desarrolladores nos cuentan qué lenguaje de programación elegirían para empezar desde cero y por qué»](#) que te ayudarán a reflexionar sobre qué lenguajes aprender, así como a comprobar la importancia concedida a Python bajo este enfoque.

Buenas prácticas en programación

Aunque en este curso no requerirás de elevados conocimientos sobre técnicas de programación, resulta interesante que consideres, desde el principio y en cualquier tarea que desarrolles en este ámbito, la importancia de desarrollar aplicaciones de «**forma estructurada, precisa y rigurosa**» (tiThink, 2018). ¿Qué quiere decir esto? Que implementes **buenas prácticas** en tus actividades de programación. ¿En qué dirías que consisten estas buenas prácticas en la programación? Ahora podrás definirla.



Buenas 19eguro19es en programación es el término genérico bajo el que se engloban un «conjunto de técnicas, principios, metodologías que debemos implementar en nuestro software para que se vuelva fácil, rápido y 19eguro de desarrollar, mantener y desplegar» (Craft Code, s.f.).

En la web Craft Code también se relacionan los principales **beneficios** asociados a la incorporación de buenas prácticas en programación. Todas estas ventajas resultan muy intuitivas y son las siguientes:

- **Código:** permite generar código del siguiente **tipo**:
 - **Limpio:** fácil de leer y escribir.
 - **Reutilizable:** que se puede emplear para la construcción de otro programa.
 - **Escalable:** sencillo de adaptar a los requerimientos de mayor uso por parte de las personas usuarias.
 - **Cambiable:** que se puede ajustar para su funcionamiento en contextos muy diferentes.

- **Colaboración:** facilita la **colaboración** con otras personas que trabajan en el mismo proyecto, dado que todas aplican una metodología común de desarrollo.
- **Errores:** permite reducir los posibles **errores de compatibilidad** entre las versiones de un programa o entre los diferentes módulos que pueden integrarlo.

Como estudiás más adelante, Python cuenta con una guía de estilo específica integrada por un conjunto de convenciones que facilitan la legibilidad del código en este lenguaje. De momento, aquí tienes una revisión de las principales **recomendaciones** planteadas para **mejorar el desarrollo del software**, recogidas en la web de tiThink (2018). Ten en cuenta que estas propuestas engloban tanto observaciones sobre el código en sí como sobre el enfoque de las actividades de programación:

- **Definir los requisitos:** es fundamental definir de manera precisa los **requisitos de la aplicación** que se desea desarrollar. Resulta muy recomendable la redacción de un documento de alcance, en el que se recojan los objetivos planteados y los mecanismos que se activarán para su consecución (como es lógico, este es un planteamiento que debes considerar en el contexto profesional).
- **Dividir en hitos:** otra de las recomendaciones es dividir la ejecución del proyecto en un conjunto de hitos, con sus correspondientes **fechas de entrega**. Es fundamental en el caso de desarrollos informáticos de amplio alcance, para así ir proporcionando a lo largo del tiempo distintos resultados parciales, que puedan ser contrastados por la clientela del servicio.
- **Emplear un Entorno de Desarrollo Integrado (IDE) adecuado:** como mínimo, que se encuentre enfocado al lenguaje de programación que se esté empleando en el desarrollo.



Como señala Kumar (2020), un IDE «es un software que ayuda en el desarrollo de software». Se trata de una aplicación que consta de una serie de herramientas, como editores de texto para escribir código fuente, mecanismos de compilación, recursos didácticos o elementos de depuración, entre otros, que facilitan de manera significativa el desarrollo de los programas.

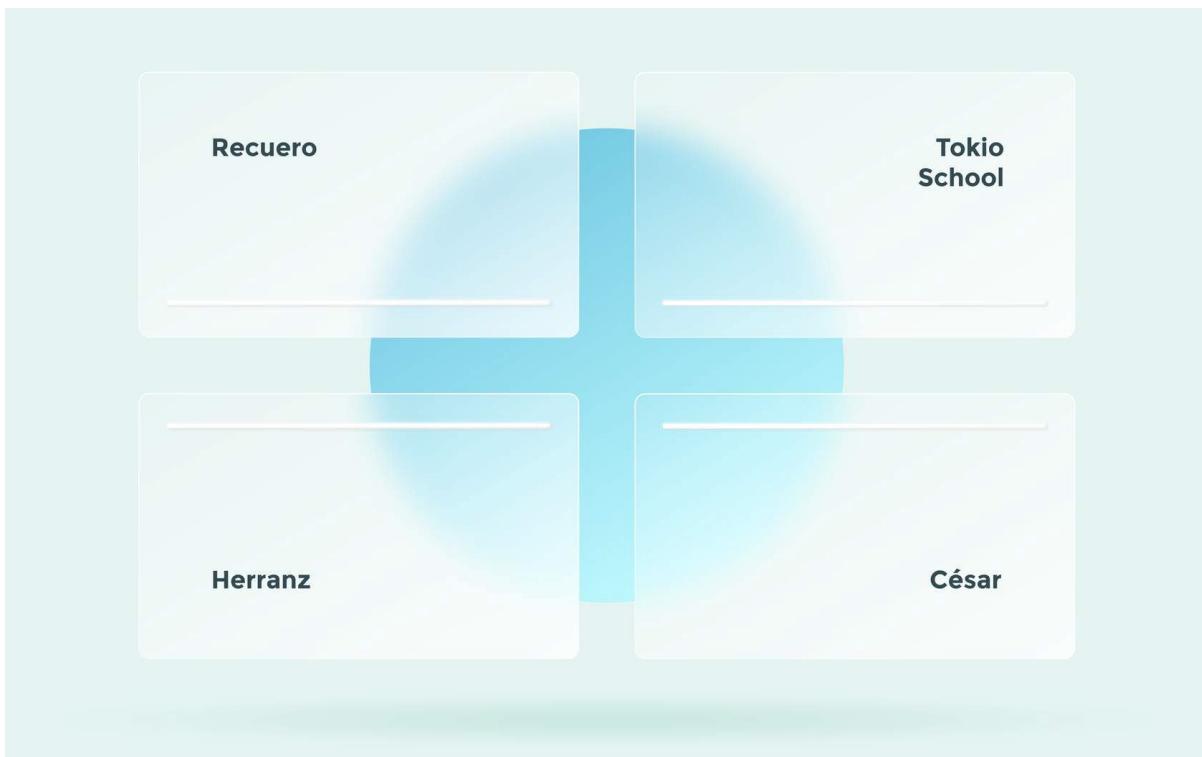
Si vas a la sección «Bibliografía», en la referencia Kumar (2020) puedes encontrar una relación de los IDE más populares que se emplean en la actualidad, lo que te permitirá comprobar las distintas funcionalidades existentes. Cuando comiences a dar tus primeros pasos en Python analizarás aplicaciones IDE específicas para trabajar con este lenguaje.

- **Estandarizar los elementos empleados en el programa:** por ejemplo, fijando el mecanismo de denominación de dichos elementos. Trabajar con **código normalizado** es una práctica excelente, dado que permite identificar rápidamente cuáles son las variables, sus atributos, las funciones, etc.
- **Aplicar el principio DRY:** *Don't repeat yourself*, es decir, **no te repitas**. La idea es que resulta muy desaconsejable repetir partes de código a lo largo de un programa. En estos casos, lo mejor es agrupar ese contenido repetitivo en una función o en otra estructura similar, a la que se llame cada vez que resulte necesario. Este planteamiento redunda en la disminución de errores y en la mejora del mantenimiento de las aplicaciones.
- **No inventar:** no desarrolles partes de aplicaciones que ya han sido satisfactoriamente redactadas por otras personas. La mayoría de los lenguajes de programación cuentan con **módulos y librerías**, verificadas por terceras personas, que se pueden adaptar y emplear en desarrollos propios, reduciendo significativamente el tiempo de trabajo.
- **Comentar el código:** resulta muy recomendable incluir observaciones (pocas, pero precisas) en las partes de código más complejas. Esto facilitará tus revisiones en el futuro y la de posibles personas colaboradoras. Este planteamiento se debe equilibrar con otro principio que señala que el mejor código es aquel que no requiere comentarios. Por lo general, has de optar por **sentencias simples y soluciones sencillas**, para así propiciar la reducción de errores.
- **Dividir desarrollos:** es conveniente seccionar desarrollos muy complejos en otros mucho más sencillos. En línea con lo propuesto en el paradigma funcional, lo idóneo es optar por un desarrollo basado en **funcionalidades específicas**, que se integren en una aplicación genérica.
- **Probar el código:** resulta fundamental incorporar mecanismos para **testar** los resultados parciales que se van alcanzando, a fin de detectar con suficiente antelación los problemas que puedan lastrar el desarrollo futuro de la aplicación.
- **Optimizar el uso de instrucciones y módulos:** de acuerdo con lo expuesto anteriormente, lo recomendable es optar en todos los casos por los **mecanismos más sencillos**.
- **Incorporar medidas de seguridad:** sobre todo en lo relativo al tratamiento de ficheros, posible desbordamiento de listas, formateo de datos de entrada, actualización de aplicaciones auxiliares (como los IDE) y empleo de contraseñas.
- **Documentar el desarrollo y las pruebas:** contar con **documentación** tanto del proceso de desarrollo como de las pruebas de la aplicación, pensando sobre todo en las posibles revisiones futuras del programa, es fundamental.

Una vez establecidas todas las cuestiones introductorias sobre los lenguajes de programación es el momento de que empieces a adentrarte en Python. ¿Crees que estás en condiciones de poder hacerlo?

Historia del lenguaje de programación Python

Antes de saber la historia de Python y cómo ha evolucionado, ¿quieres comenzar por conocer algunas **opiniones** de personas expertas en la materia sobre este lenguaje:

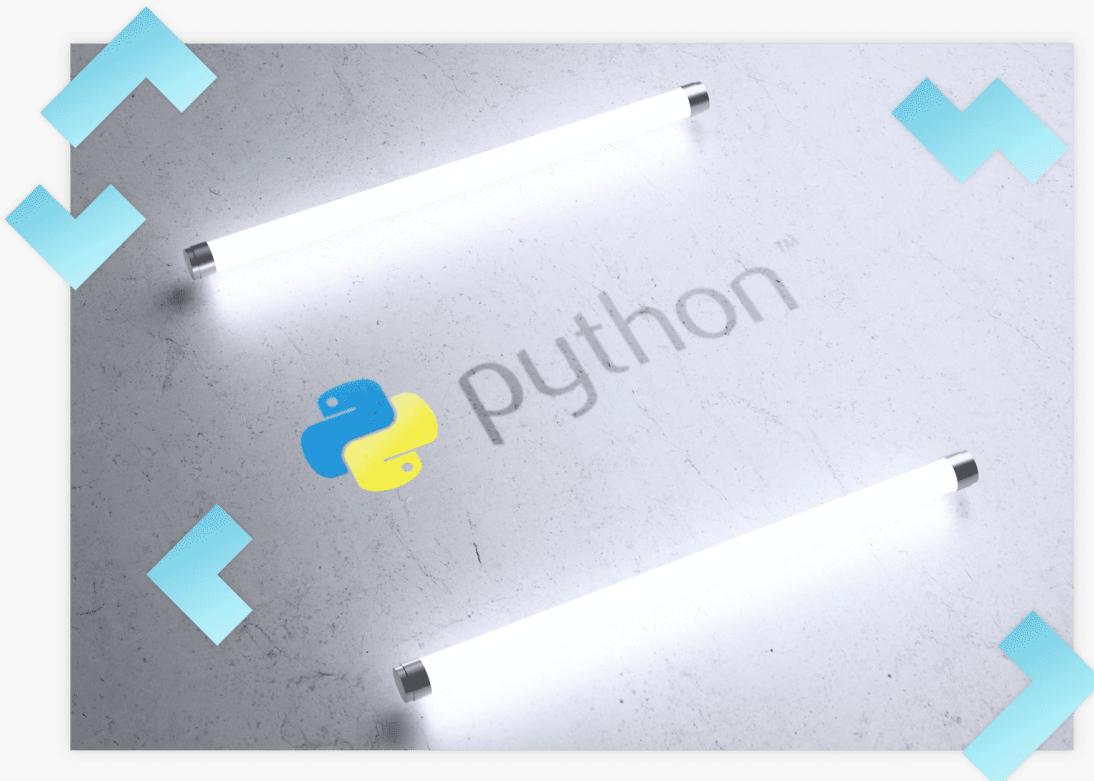


- **Recuero:** según Recuero (2020), «Python es, en estos momentos, el lenguaje más popular». Esta afirmación se basa en el análisis del volumen y grado de interés suscitado por las preguntas formuladas sobre dicho lenguaje en páginas web como StackOverflow (un repositorio enfocado a profesionales de la programación, en la que se recogen preguntas y respuestas sobre estas materias).
- **Tokio School:** una idea similar se recoge en la web de Tokio School (2020), en la que se indica que «Python es el tercer lenguaje de programación más usado en el mundo».
- **Herranz:** Herranz (2021) cuantifica dicha expansión y estima que en el mundo trabajan 10,1 millones de personas dedicadas al desarrollo de este lenguaje (según el informe en el que se basa este dato, JavaScript sería el que cuenta con un mayor número de personas desarrolladoras, en torno a 13,8 millones).
- **César:** como describe César (2020), Python es utilizado por empresas de alta tecnología, de sectores muy diferentes, como Google (continúa empleándose en parte del motor de búsqueda), Instagram, Facebook, Spotify, Amazon (en lo referente a la generación de sugerencias en base a los patrones de compra de las personas usuarias) o Stripe (que es una plataforma Fintech especializada en pagos *online*) entre otras.

Seguro que todos estos datos te estarán generando una pregunta: ¿cuál es la razón del **éxito** de este lenguaje de programación y de su preeminencia en un contexto en el que existen tantas herramientas alternativas? A medida que profundices en el conocimiento de Python podrás confirmar su potencial y su elevado nivel de excelencia, sobre todo para trabajar en el campo de la *Data Science* (o ciencia de los datos).



¿Eres consciente de que la traducción literal, del inglés, del término «Python» es pitón? Por eso, en las portadas de libros o en los iconos de las páginas web focalizadas en este lenguaje es muy común que aparezcan representaciones de serpientes, generalmente vinculadas a algún motivo tecnológico. De hecho, el logotipo oficial de Python (Python, s.f.), que tienes reproducido a continuación, incorpora dos serpientes, una de color azul y la otra, invertida, de color amarillo, inspiradas en antiguas representaciones mayas.



Sin embargo, la denominación de Python, a cargo de su desarrollador, Guido Van Rossum, obedece a otra motivación, y es su admiración por el grupo británico de humor Monty Python, autores de una serie célebre de televisión, *Monty Python's Flying Circus*, y de películas de referencia como *La vida de Brian* o *El sentido de la vida*, entre otras.

Ahora te centrarás en la historia y evolución de Python.

Como seguramente sabrás, prácticamente todos los lenguajes de programación evolucionan con el tiempo, a fin de adaptarse a los **nuevos requerimientos tecnológicos** o para incorporar **nuevas funcionalidades**, normalmente demandadas por las personas usuarias.

La **primera versión pública de Python**, a cargo de Van Rossum, se lanzó en febrero de 1991, aunque su autor venía trabajando en este lenguaje desde finales de 1989.

La web de Tokio School (2020) señala que esta primera versión, la **0.9.0**, ya se caracterizaba por su facilidad de empleo por parte de individuos con escasos conocimientos de programación, incorporando un aspecto básico de este lenguaje, el **enfoque modular**. A partir de ese momento un equipo de personas desarrolladoras, dirigido hasta 2018 por Guido Van Rossum, comenzó a trabajar en modificaciones de Python, lo que se ha ido concretando en el lanzamiento público de numerosas versiones. Como se indica en la web de MCLIBRE (s.f.), desde 2019, «el desarrollo de Python está dirigido por un consejo de dirección de cinco miembros», que se renueva anualmente.



Acabas de comprobar que una versión cualquiera de Python se encuentra identificada por **tres dígitos** (que denotaremos a.b.c). ¿Qué significa cada uno? Descúbrelo a continuación:

- **Primer dígito (a):** identifica a una **gran versión** del lenguaje. Hasta el momento existen 4 grandes versiones, que resultan incompatibles entre sí, especialmente a partir de la 3. Un programa escrito en código de la versión 2 no funcionará, por lo general, o no lo hará correctamente, en la versión 3 y viceversa; se requerirá modificar aspectos de la programación para garantizar un correcto desempeño. Estas son las **cuatro grandes versiones** existentes hasta el momento:
 - **Versión 0:** fue la primera que se publicó, todavía bajo propiedad del centro de investigación holandés (CWI) el que trabajaba Guido Van Rossum.
 - **Versión 1:** se lanzó en 1994. La desvinculación con el CWI tuvo lugar con la publicación de la versión 1.2. Las principales novedades de estas variantes se correspondieron con la incorporación de herramientas que facilitaban el paradigma funcional.
 - **Versión 2:** vio la luz en octubre del año 2000. Las modificaciones más relevantes se centraron en el tratamiento de las listas, que es otro de los elementos más característicos de Python. El recolector de basura para referencias cíclicas también se introdujo en este contexto.
 - **Versión 3:** la última gran versión del lenguaje es la 3, publicada en 2008, con importantes cambios en la sintaxis y en la tipología de datos, focalizándose en la supresión de la duplicidad de elementos. Introdujo, por ejemplo, el tratamiento de print como una función, lo que obliga desde entonces al uso de paréntesis.

No existen, en la actualidad, rumores sobre un posible lanzamiento de una nueva gran versión, que sería, como es evidente, la 4.

- **Segundo dígito (b):** sirve para referenciar versiones con **cambios relevantes** en el lenguaje, pero que mantienen la compatibilidad entre sí, por ejemplo, código escrito en la versión 3.4 debe funcionar perfectamente en la 3.7, aunque existen algunas excepciones muy puntuales, recogidas en las fuentes oficiales del lenguaje. La web de MCLIBRE (s.f.) señala que estas versiones intermedias a.b se suelen publicar cada año y medio aproximadamente, aunque no existe un período fijado para ello.
- **Tercer dígito (c):** corresponden a **versiones menores**, que dependen de la a.b ya publicada, que incorporan medidas para la corrección de errores y problemáticas de seguridad.

En el momento de la redacción de este curso, la última versión estable es la **3.9.7**, con mejoras en el tratamiento de diccionarios, cadenas de texto, tipado de funciones, etc. Ten en cuenta que las versiones irán cambiando con el tiempo, ya que van evolucionando, pero el contenido explicado se aplica a las versiones posteriores.

Resulta recomendable tener siempre instalada en el equipo la última versión estable, aunque esta afirmación es matizable, debido a posibles problemas de compatibilidad con lo que se denominan **librerías**. También es factible contar con distintas versiones en el mismo soporte. Esto resulta útil en el caso de disponer de aplicaciones antiguas que, por ejemplo, no funcionan en versiones modernas del lenguaje.



Desde el lanzamiento de la versión 1 se encuentra operativo un grupo oficial de noticias denominado **comp.lang.python**, al que puedes acceder y unirte, por ejemplo, a través de un correo Gmail (debes acceder a la sección de Grupos y buscar el indicado). También existe una lista oficial de correos, **python-list**, interconectada con el grupo de noticias. El volumen de información que se genera en comp.lang.python es muy elevado, pero este grupo constituye un recurso excelente para estar al día sobre las últimas novedades del lenguaje o localizar avances y desarrollos en materias de programación en Python que resulten de tu interés.

Características del lenguaje Python

Para comprender cuáles son los aspectos que confieren tanta relevancia en la actualidad a Python resulta apropiado partir de la siguiente definición.



Python «es un lenguaje de programación interpretado, de código abierto, multiparadigma, aunque principalmente orientado a objetos, de alto nivel. Su sintaxis hace énfasis en la legibilidad del código, lo que facilita su depuración y, por tanto, favorece la productividad. Ofrece la potencia y la flexibilidad de los lenguajes compilados con una curva de aprendizaje suave» (Recuero, 2020).

Ya has estudiado algunas de las **características** recogidas en la definición anterior, aunque ahora podrás ampliar esta visión y concretarlas para el lenguaje que estás comenzando a estudiar. Seguiremos para ello la revisión propuesta por Recuero (2020), junto a aportaciones de otras publicaciones:

- **Lenguaje interpretado:** ya sabes que los programas escritos en **lenguaje interpretado** tienen una velocidad de ejecución más lenta que el caso de que fueran compilados. Esta desventaja se está reduciendo en la actualidad gracias a los procedimientos más recientes de computación en la nube. Recuerda, además, que existen otros factores positivos asociados a esta tipología de lenguajes, como una mejor adaptación a los entornos multiplataforma o una mayor velocidad de desarrollo.
- **Propósito general:** otro aspecto reseñable de Python es que se encuentra concebido con un **propósito general**, es decir, se puede emplear para crear aplicaciones en todo tipo de contextos. En la web El Pythonista (s.f.) se destacan las siguientes **áreas**:
 - Ciencia e ingeniería.
 - Análisis de datos, Machine Learning, *Data Mining*, Visión Artificial... En ese sentido, Recuera (2020) señala que se trata del lenguaje preferido por un 57 % de las personas especializadas en el tratamiento de datos.
 - Desarrollo de aplicaciones web.
 - Aplicaciones de escritorio (en especial para el sistema operativo Linux).
 - Desarrollo de videojuegos.
 - Ciberseguridad.
 - Finanzas, con desarrollos específicos para el tratamiento de criptomonedas, trading y creación de gráficas.
 - Administración de sistemas y DevOps.
 - Desarrollo de *scriptings*, que son pequeños programas que se integran en aplicaciones de mayor tamaño (en este curso emplearemos, de manera indistinta, programa y *script*, dado que, por lo común, nuestros desarrollos no serán de gran tamaño).



Python es uno de los lenguajes más empleado para el desarrollo de aplicaciones por parte de las cinco empresas que constituyen el GAFAN. Bajo este acrónimo (en inglés el término que se emplea es FAANG) se engloban las grandes empresas tecnológicas que cotizan en el Nasdaq (el índice bursátil de referencia en EE. UU.): Google, Amazon, Facebook, Apple y Netflix. Esto refuerza la idea del potencial generalista que caracteriza a dicho lenguaje.

Python también se emplea en el análisis de textos (*Text Mining*), para la elaboración de estadísticas, para la gestión de recursos telemáticos, etc. Esta amplia versatilidad del lenguaje se fundamenta, además de en su diseño, en la existencia de numerosas **librerías y frameworks**, que facilitan la especialización en distintas temáticas.

- **Compatibilidad:** este lenguaje es plenamente **compatible** con otros, en especial C, Java o R, lo que facilita la implementación en proyectos complejos o en el desarrollo de los ya existentes, desarrollados con lenguajes diferentes.
- **Legibilidad:** la sintaxis de Python resulta muy próxima a la que se emplea en el lenguaje humano (esto lo irás comprobando a medida que revises y generes código). Esta **legibilidad** tan relevante (basada en el empleo de la indentación) es la que posibilita que el aprendizaje se realice de manera natural, con poco esfuerzo (de ahí que se emplee la expresión «curva de aprendizaje suave»). En la web El Pythonista (s.f.) se emplean ciertas dosis de humor (muy en la línea de los Monty Python) para señalar que Python «permite escribir código como si se tratase de una receta en inglés, más que un lenguaje de programación».
- **Reglas de estilo:** el empleo de **reglas de estilo**, disponibles para todas las personas usuarias, posibilita que el código creado sea muy consistente y unificado. La existencia de dichas reglas, así como la concepción genérica del lenguaje, facilitan la creación de los programas de manera muy rápida.
- **Paradigmas:** Python puede adoptar diferentes **paradigmas** de programación, en función de los objetivos que se deseen alcanzar. La metodología que, normalmente, se suele emplear es la orientada a objetos, aunque también se puede adaptar para trabajar en el contexto funcional y en el imperativo, sobre todo bajo los enfoques estructurados y modulares.
- **Entornos educativos:** Python es uno de los lenguajes más empleados en los **entornos educativos**, con el objetivo de formar a futuras personas expertas en programación. Esto se debe tanto a su facilidad de empleo como por ser multiplataforma, lo que reduce la necesidad de instalar software adicional en los equipos ubicados en las aulas.

- **Software libre:** Python se inscribe dentro de la corriente de **software libre (free software)**, lo que «significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software», según la definición recogida en la web de GNU (s.f.). Es importante que tengas en cuenta que esta libertad que se otorga a las personas integrantes de una comunidad no implica que dicho software tenga que ser gratuito (aunque este sea el caso de Python).

Para que un lenguaje de programación o una aplicación se considere software libre resulta necesario que se cumplan **cuatro principios**, recogidos en la web de GNU (s.f.):

- **Principio de libertad 0:** las personas usuarias han de ser libres para emplear el software de la manera que deseen, con el propósito que estimen oportuno.
- **Principio de libertad 1:** debe resultar posible el estudio del funcionamiento del software, así como la introducción de cambios y modificaciones para que haga lo que se desee. Esto requiere el acceso al código fuente. En el caso de Python, se encuentra accesible en todo momento a través del enlace de descarga de Phyton, incluso resulta posible consultar el código fuente que se está desarrollando)
- **Principio de libertad 2:** las personas usuarias han de poder distribuir copias del software original, con el objetivo de ayudar a otros individuos.
- **Principio de libertad 3:** como ampliación de la idea anterior, también resultará factible la posibilidad de mejorar el software y la difusión de dichas modificaciones positivas entre los integrantes de la comunidad.

En Recuero (2020) se destaca que Python cumple, a partir de la versión 2.1.1, con los estándares recogidos en la Licencia Pública General de GNU, que es la licencia de uso de software más extendida en este ámbito. La publicación en sí del lenguaje se realiza bajo la licencia de la Python Software Foundation, una organización sin ánimo de lucro que gestiona el desarrollo integral del proyecto.

- **Código abierto:** Python también se caracteriza por ser un lenguaje de **código abierto (open source)**. Es posible que pienses que se trata de lo mismo que free software, pero existe una importante diferencia a nivel conceptual. El open source defiende que no debe existir software privativo o propietario (con código al que no se puede acceder de manera pública) considerando **aspectos técnicos**, dado que la solución óptima es que quienes integran la comunidad puedan aportar sus conocimientos y mejorar de manera conjunta los desarrollos existentes. El movimiento *free software*, por su parte, se basa en **condicionantes éticos**, reclamando una sociedad en la que se garantice la libertad de las personas usuarias, de acuerdo con los cuatro principios descritos.

El hecho de que Python sea libre y abierto ha propiciado el desarrollo de una **comunidad** en expansión, muy activa, que con sus observaciones e investigaciones ha contribuido a un desarrollo en profundidad del código de este lenguaje.

Primeros pasos en Python: antes de comenzar

Para comenzar a trabajar con Python es recomendable tener instalado en tu equipo un **entorno de programación**, que variará ligeramente en función del sistema operativo con el que cuentes. También es posible operar en **entornos virtuales**, así no tendrás que preocuparte de la implementación y configuración del lenguaje. Sin embargo, dados los condicionantes de las técnicas de Machine Learning, lo más razonable será que, en su momento, instales herramientas que te permitan **programar en local o en una red interna de equipos**.

Lo recomendable es emplear la **última versión estable del lenguaje**. ¿Cómo puedes saber cuál es? Accede a la web oficial de [Python](#) y pasa el ratón por encima de la sección Downloads. De manera inmediata te aparecerá la información asociada a Windows.

Ten la precaución de utilizar siempre la **última versión estable** y comprueba que el sistema no te alerte de posibles incompatibilidades con tu Sistema Operativo. La descarga es inmediata, basta con hacer clic en el recuadro en gris para disponer del fichero ejecutable en la carpeta que tengas habilitada al respecto.

Si tu sistema operativo no es Windows (o su versión no admite la última versión estable) bastará con hacer clic en la opción "macOS", en "Other Platforms" o, más simple, en "View the full list of downloads", con lo que dispondrás del enlace a todas las versiones existentes para las distintas plataformas. También haciendo clic en los enlaces recogidos en la parte superior de esta misma página accederás a las últimas versiones de Python para Linux/UNIX, para y para otras plataformas (AIX, IBMi, iOS, Solaris...).

Otra manera de acceder a la parte de descargas de Python, además de la que acabas de conocer, es yendo, directamente a la sección de [Downloads](#). Aquí encontrarás una tabla en la que se ofrece información sobre versiones recientes o relevantes, indicando cuál es el estado de mantenimiento. Como es lógico, los vínculos a versiones antiguas tienen interés para personas especializadas en desarrollos, que necesiten continuar trabajando con aplicaciones basadas en dichas modelos.

Expresándolo de manera muy simple resulta posible diferenciar **dos enfoques genéricos** a la hora de proceder a la **implementación de un entorno de programación en Python** (el sistema operativo con el que estés trabajando influirá en la solución definitiva que puedas alcanzar). Son los siguientes:

- **Personas principiantes:** realizar una instalación enfocada a personas principiantes, con **nula o poca experiencia** en Python. En este caso priorizarás que la implementación resulte sencilla y te centrarás en la activación de las **funcionalidades más comunes** del lenguaje. Lo más recomendable es instalar el paquete que corresponda de la web oficial de Python.



- **Distribuciones de Python:** la otra opción es acudir a lo que se conoce como distribuciones de Python, especialmente enfocadas a ciertos **ámbitos de trabajo**, lo que facilita una implementación más profesional, que facilita a personas expertas el desarrollo de **tareas con un mayor nivel de complejidad** tanto desde el punto de vista de puesta en producción de aplicaciones como para el desarrollo de investigaciones.

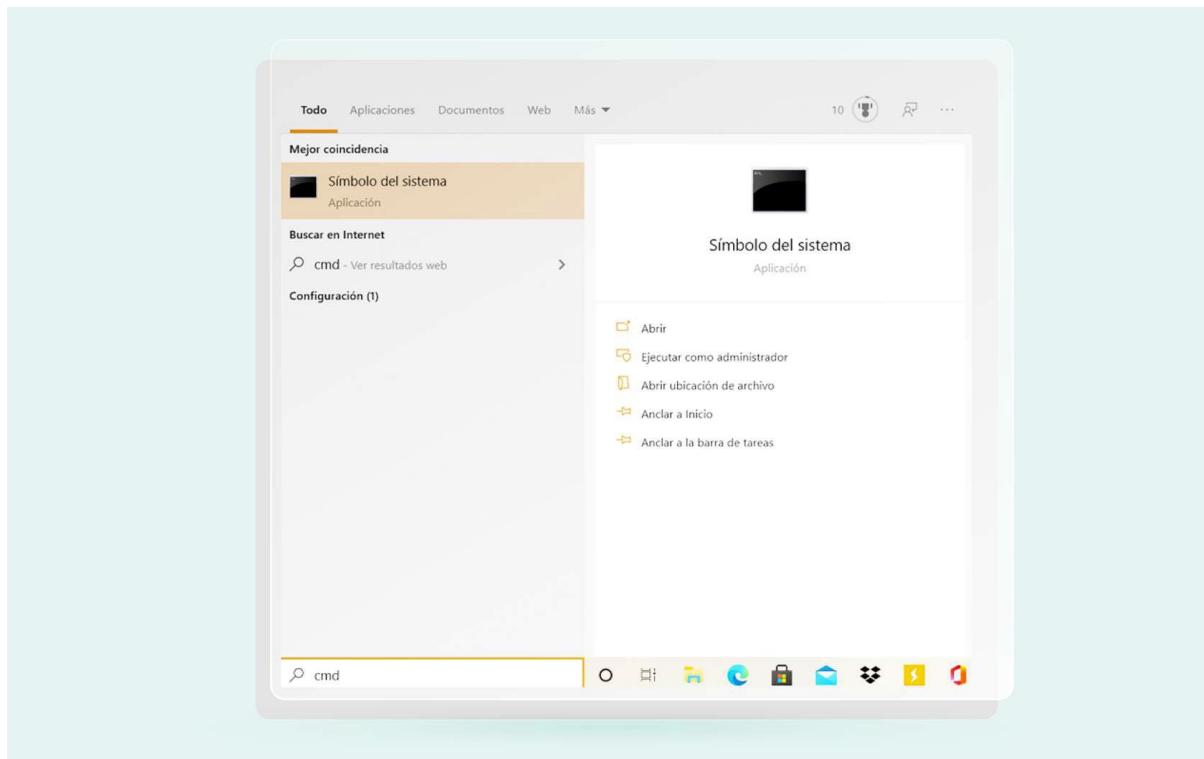


En esta primera parte del curso nos centraremos en el **mecanismo de implementación más simple**, aunque, de manera paulatina, te irás adentrando en otras distribuciones de interés, entre las que destaca Anaconda, al estar orientado a desarrollos en el área de Machine Learning.

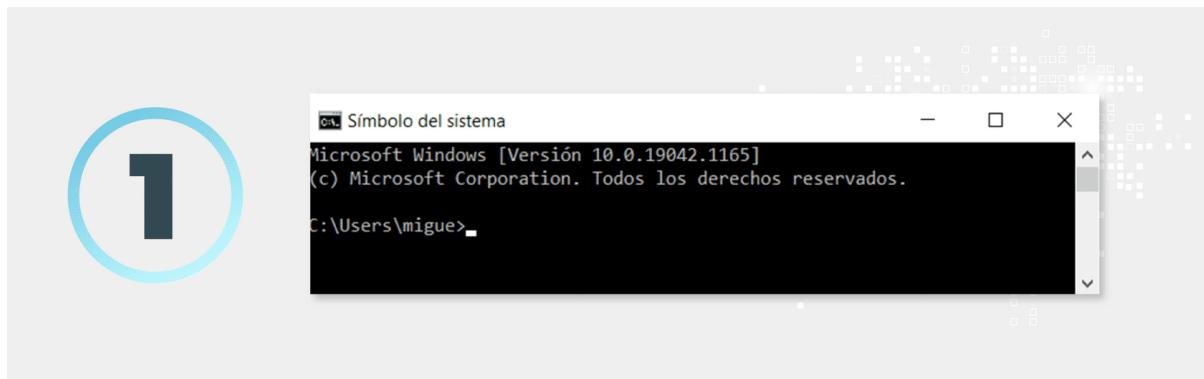
Primeros pasos en Python: comprobación versión existente

Una instalación sencilla en **Windows**, que es el sistema operativo más extendido en la actualidad, debe comenzar comprobando si existe alguna versión previa de Python en el equipo. ¿Sabes cómo hacerlo? Hay varias opciones.

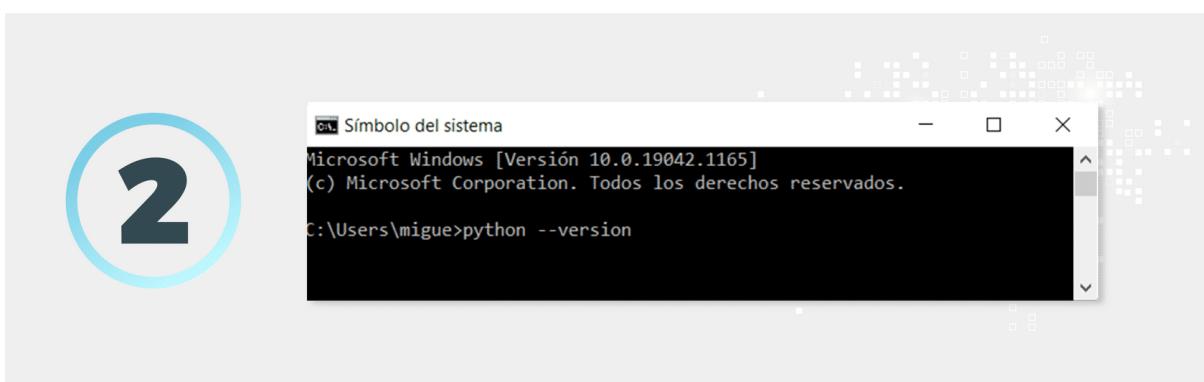
Lo más fácil es teclear **cmd** en la barra de búsqueda (normalmente, en la esquina inferior izquierda, lo que hará que se muestre la ventana de Símbolo del sistema).



Otra opción consiste en pulsar simultáneamente la tecla con el logo de Windows y la tecla R y en la caja que aparece en pantalla escribir **cmd**. Al pulsar intro (o al pinchar en la opción Abrir de la imagen anterior o en el propio Símbolo del sistema) aparecerá una interfaz similar a la que se muestra a continuación:

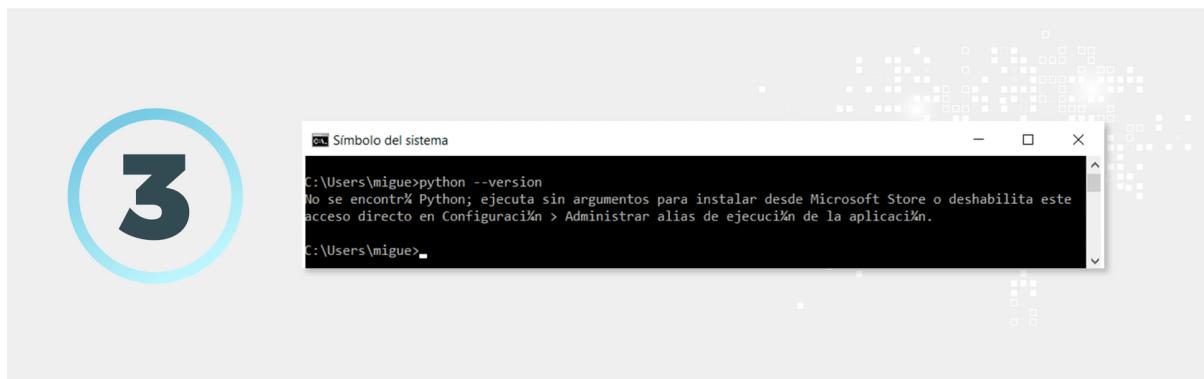


Bastará con escribir en la pantalla la sentencia `python --versión` y pulsar intro.



Si hubiera alguna versión ya instalada, el sistema informaría de su existencia e indicaría de cuál se trata. En tal caso, tendrías que valorar si mantenerla o desinstalarla y emplear la última versión estable, de acuerdo con las observaciones realizadas anteriormente.

En caso de que no hubiera ninguna versión instalada se obtendría un mensaje similar al que se ofrece en la siguiente imagen (o, simplemente, un mensaje de error):





No resulta infrecuente la instalación «inconsciente» de Python en los equipos informáticos. En ocasiones instalamos aplicaciones, sobre todo de orientación estadística o matemática, como SPSS (un paquete estadístico desarrollado por IBM) o Matlab (a cargo de MathWorks), que ofrecen la posibilidad de implementar pequeños programas o *plugins* que permiten interconectar dicho software con otras herramientas muy populares, caso de Python o R. En estos casos, al dar la conformidad a dicha opción se procede a la instalación, de manera transparente a la persona usuaria, de una 32erisión de Python y del correspondiente *plugin*.

Primeros pasos en Python: instalación Windows

Una vez adoptada la decisión de **implementar la última versión de Python** cuentas **con dos opciones:** Microsoft Store o instalación directa. Ahora sabrás más sobre cada una de ellas.



La más sencilla consiste en abrir la aplicación Microsoft Store, que en Windows 10 suele encontrarse en la barra de herramientas (también la puedes localizar a través de la caja de búsqueda). Una vez abierta, en el apartado superior de búsqueda teclearás **python**, de modo que el sistema te ofrecerá el conjunto de versiones disponibles.

En este caso, deberás pulsar el ícono de la última versión estable (que también podrás conocer con una simple búsqueda en Internet). La instalación a partir de este momento es prácticamente inmediata y, aunque implementa un intérprete de Python muy básico, tiene algunas **ventajas**, como se recoge en la web oficial de Microsoft (2021):

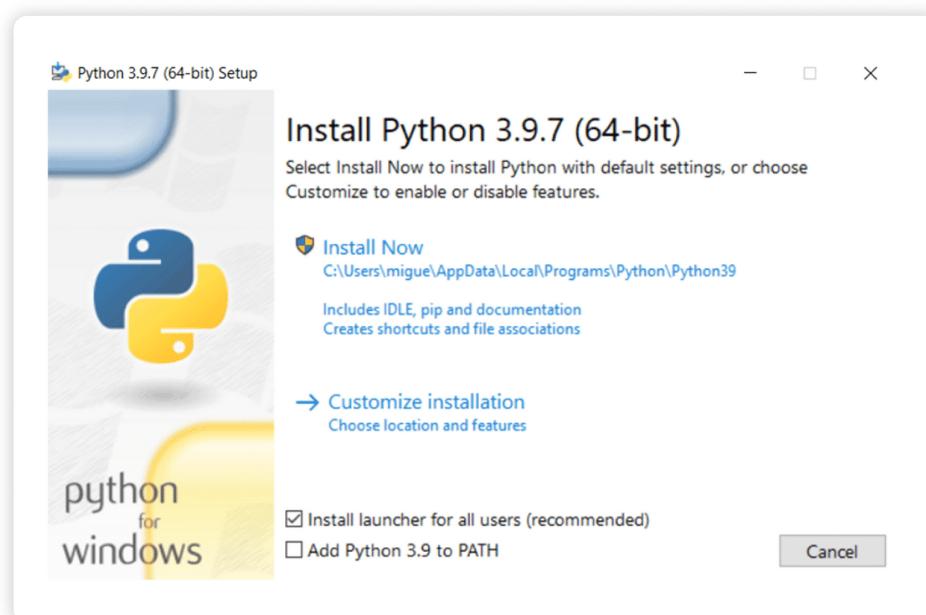
- Permite configurar correctamente el valor **PATH** para la persona usuaria actual, sin requerir un acceso de rol administrador (**PATH** hace referencia, básicamente, a una variable del sistema operativo en la que se especifican las rutas de búsqueda de los intérpretes de los comandos).
- Se aseguran las **actualizaciones automáticas** del lenguaje, sin necesidad de que la persona usuaria esté revisando la posible aparición de nuevas versiones.

- Es una instalación idónea para **actividades formativas** centradas en Python.
- También es muy útil para organizaciones que tengan implementados sistemas de **control administrativo** de acceso a aplicaciones o a equipos (precisamente porque no requiere atributos de administrador para la instalación).



La segunda alternativa, que es la recomendable en esta parte del curso, consiste en **instalar directamente**, como cualquier otra aplicación, la última versión estable recogida en la web de Python:

Paso 1: Descarga



Bastará que, por ejemplo, hagas clic en el recuadro debajo de Download for Windows, tal y como viste en una imagen anterior. Instalándolo de este modo dispondrás en tu equipo del ejecutable del lenguaje. Solo necesitarás abrirlo y te saldrá una pantalla como la de la imagen superior.

Fíjate en que resulta posible personalizar la instalación (entrando en Customize installation), aunque es preferible que no realices cambios, ya que la configuración por defecto incluye documentación informativa, un sistema de gestión de paquetes de software escritos en Python denominado pip, el entorno de desarrollo IDLE, una librería para la realización de pruebas y un lanzador específico del lenguaje en Windows (lo que acelera su arranque).

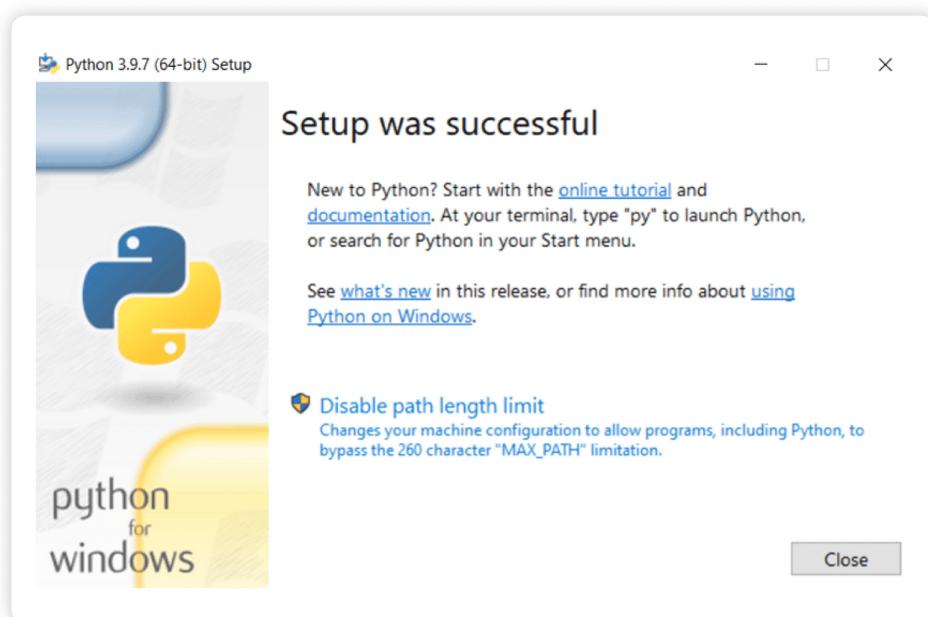
Paso 2: Incorporar Python al PATH



Lo recomendable es que valores si realizas una instalación para todas las personas usuarias (dependerá de quiénes y de qué manera usan el equipo) o desmarcas esa opción. Lo que sí resulta muy aconsejable es **marcar la última opción**, la de incorporar Python al PATH, dado que esto facilitará la **correcta configuración del sistema**. Por tanto, deberías iniciar tu instalación desde una pantalla como esta que muestra la imagen superior (considerando, por ejemplo, que lo haces en un equipo propio, que únicamente empleas tú).

Una vez que pulses en **Install Now** arrancará la **instalación** en sí, cuya duración dependerá de la capacidad de tu equipo, aunque no debe exceder del minuto o minuto y medio.

Paso 3: Confirmación



La pantalla final, que en principio confirma que el proceso se ha desarrollado correctamente, es la que tienes en la imagen superior.



Seguramente te estarás preguntando por la opción de deshabilitar el **límite de longitud de la ruta** (*Disable path length limit*), que aparece al final del mensaje. Windows tiene una limitación, de unos 260 caracteres, en lo que respecta a la ruta de localización de cualquier archivo que, como sabes, se encuentra formada por el nombre y todas las carpetas que lo contienen. Dado que uno de los objetivos centrales de Python es que funcione correctamente en todas las plataformas, el lenguaje incorpora la opción de deshabilitar dicha limitación (en Windows 10 eso también se puede hacer actuando en la configuración de la directiva del equipo local). Otra alternativa consiste, simplemente, en tener cuenta esta restricción y trabajar siempre con rutas con un tamaño por debajo del máximo. Puedes saber más sobre estos condicionantes en un blog de DelfStack, si haces clic en el siguiente enlace y accedes al artículo [«Deshabilitar el límite de longitud de la ruta en Python»](#).

Paso 4: Cerrar



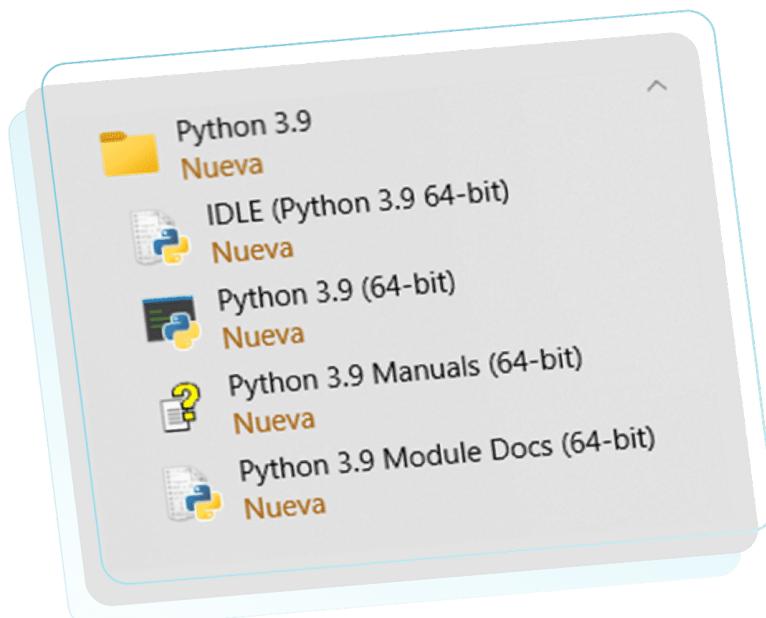
```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19042.1165]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\migue>python --version
Python 3.9.7

C:\Users\migue>
```

Tras valorar la idoneidad o no de deshabilitar el límite de longitud de la ruta (en muy raras ocasiones te resultará necesario hacerlo), harás clic en Close, con lo cual **concluirás la instalación**. Activando de nuevo el símbolo del sistema y tecleando la misma sentencia que antes (**cmd**) comprobarás que, en efecto, se ha implementado la versión seleccionada como te muestra la imagen superior.

Si acudes a la carpeta de Python que se habrá generado en la sección de inicio de Windows (haciendo clic en el logo de Windows que hay en la esquina inferior izquierda) podrás comprobar cuáles son las **aplicaciones y documentos incorporados** tal y como muestra la siguiente imagen:



Primeros pasos en Python: instalación macOS y Linux

Ya sabes cómo instalar Python en Windows, el sistema operativo más extendido. Ahora descubrirás cómo hacerlo con **macOS y Linux** que, aunque sean menos empleados, también debes conocer cómo proceder.



macOS

La operativa de instalación de Python en **macOS** es muy similar. Para comprobar si ya existe alguna versión previa de este lenguaje debes acceder a **Finder/Aplicaciones/Utilidades/Terminal**. En la ventana que se abrirá, donde te permite incorporar un texto (tras el signo \$), escribe **python**, en este caso con especial atención a que esté todo en minúsculas (aparte de indicarte la versión, este comando abre el programa intérprete para poder añadir instrucciones). Resulta habitual que este sistema operativo ya traiga incorporada una versión de Python, el problema es que suele encontrarse obsoleta.

En el equipo en el que hemos hecho la prueba la respuesta ha sido la siguiente:

```
$ python
```

WARNING: Python 2.7 is not recommended.

This version is included in macOS for compatibility with legacy software.

Future versions of macOS will not include Python 2.7.

Instead, it is recommended that you transition to using 'python3' from within Terminal.

Python 2.7.16 (default, Jun 18 2021, 03:23:53)

```
[GCC Apple LLVM 12.0.5 (clang-1205.0.19.59.6) [+internal-os, ptrauth-is=deploy on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

Fíjate en que el sistema te recomienda la instalación de una versión posterior del lenguaje, aun cuando te permite, en la última línea, introducir comandos de Python, justo detrás de los signos `>>>`, que revelan que se encuentra activado el intérprete. En el caso de disponer de una versión 3. del lenguaje, la llamada a través del terminal se tendría que realizar mediante la instrucción `python3` (si escribieras solo `python` el sistema activaría la versión 2).

Para implementar la última versión de Python únicamente tienes que entrar en la web oficial, descargar la que corresponde al sistema operativo y lanzar la instalación, como cualquier otra aplicación. Una vez que el proceso haya concluido, pon en marcha de nuevo el terminal y escribe `$python3 -version`, para así confirmar la idoneidad de la versión.



Linux

Finalmente, en **Linux** lo habitual será disponer, de partida, de una versión bastante reciente del lenguaje, y bastará con realizar algunos ajustes sencillos. Tanto para este sistema operativo como para otros de uso más residual, es recomendable que consultes las instrucciones de instalación, muy clarificadoras, recogidas en la web de Python.

Primeros pasos en Python: mensaje de bienvenida al mundo

Para iniciar tus primeros pasos en Python es recomendable que trabajes sobre el **programa de base** desde el que parten la mayoría de las personas que realizan labores formativas en el ámbito informático: la aparición en pantalla del texto **¡Hola mundo!** (el célebre programa "Hello world!" del que ya viste dos ejemplos anteriormente).

¿Por qué? Porque resulta sencillo y, a la vez, sirve para confirmar el correcto funcionamiento de la instalación realizada. Por lo general, si no hay problemas en su ejecución es que tampoco existen en su implementación. Algunos individuos sostienen, además, que comenzar con este programa proporciona suerte, así que ¡manos a la obra!



Hay páginas como helloworld.org en la que se recoge el código asociado a este programa en diferentes lenguajes de programación. Accede a este contenido y analiza las similitudes y diferencias detectadas.

Una vez que estás aquí, ¿sabes cómo escribir código en Python? Existen diferentes maneras de hacerlo. Conocerás estas **tres alternativas de escribir código en Python**, cada una de utilidad en función del objetivo que te estés planteando:

- Lanzar directamente el intérprete de Python.
- Emplear un editor de texto.
- Acudir a la ayuda de un IDE.

Lanzar directamente el intérprete de Python

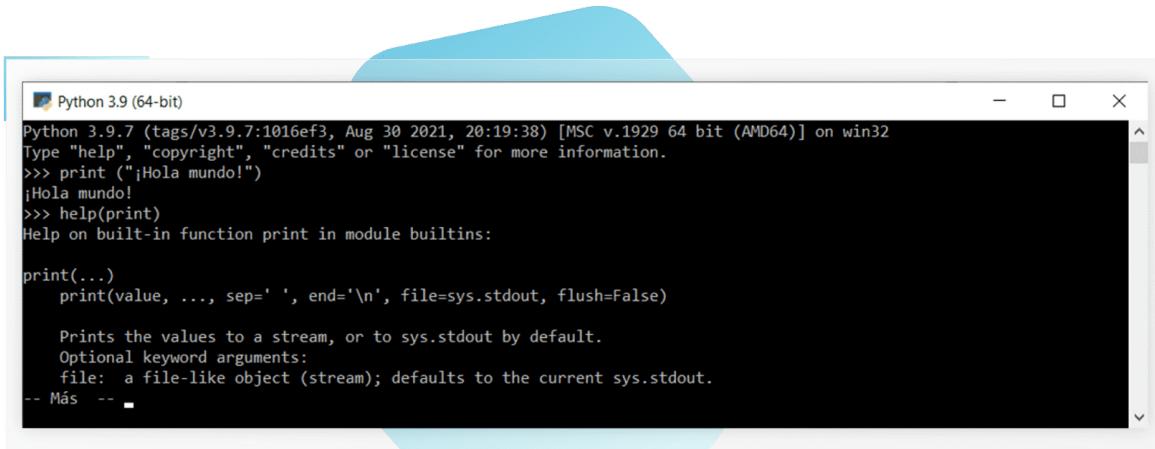
La **primera alternativa** y la más inmediata de escribir código en Python consiste en **lanzar directamente el intérprete de Python**, que es la aplicación que aparece denotada como Python 3.9 en la carpeta de aplicaciones de dicho lenguaje, tal y como muestra la imagen:



Al hacer clic en el programa se abre la **terminal**, con los signos >>> indicando que se puede proceder a la escritura de comandos. En este caso introduciremos print ("¡Hola mundo!") y pulsaremos intro, obteniendo el resultado que muestra la imagen:

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("¡Hola mundo!")
¡Hola mundo!
>>>
```

Si quisieras **cerrar la aplicación** bastaría con pinchar en el aspa (ángulo superior derecha) o simplemente escribir `quit()` o `exit()`. Fíjate en que si desearas obtener información inmediata de `print` (que es una función muy útil de Python) únicamente tendrías que escribir `help(print)` y pulsar intro.



En este curso emplearemos la terminal únicamente para hacer **pequeñas pruebas de sintaxis** y comprobar algunos **comportamientos de variables**, pero no será lo habitual, dado que este entorno se encuentra muy limitado para nuestros desarrollos futuros.

Emplear un editor de texto

Una **segunda alternativa**, más funcional, consiste en emplear un **editor de texto**, que, simplemente, es una aplicación que permite crear y editar texto sin formato. Esto es fundamental, dado que los lenguajes de programación requieren trabajar con texto plano. Si, por ejemplo, elaboras un documento con Microsoft Word -que es un **procesador de texto**- o similar, el programa incorporará una información, de manera transparente para la persona usuaria, sobre el formato del contenido trabajado, lo que puede provocar problemas si se emplea como código de programación.

Como se recoge en Martínez (2021), los editores de texto poseen una serie de **utilidades**, que les dotan de gran atractivo para la redacción de los códigos. Fíjate en la imagen para conocerlas:

Cuentan con funciones de corte, pegado, importación, modificación, retroceso de acciones, etc. similares a las de los procesadores de texto.



Disponen de opciones de búsqueda y reemplazo de cadenas de texto y numéricas.

Permiten resaltar la sintaxis empleada (normalmente, a través de códigos de colores), así como aplicar autocompletados y visualizar código mediante diferentes pestañas.



También posibilitan emplear listados y bases de datos, por lo común a través de gestores especializados en lenguaje SQL.



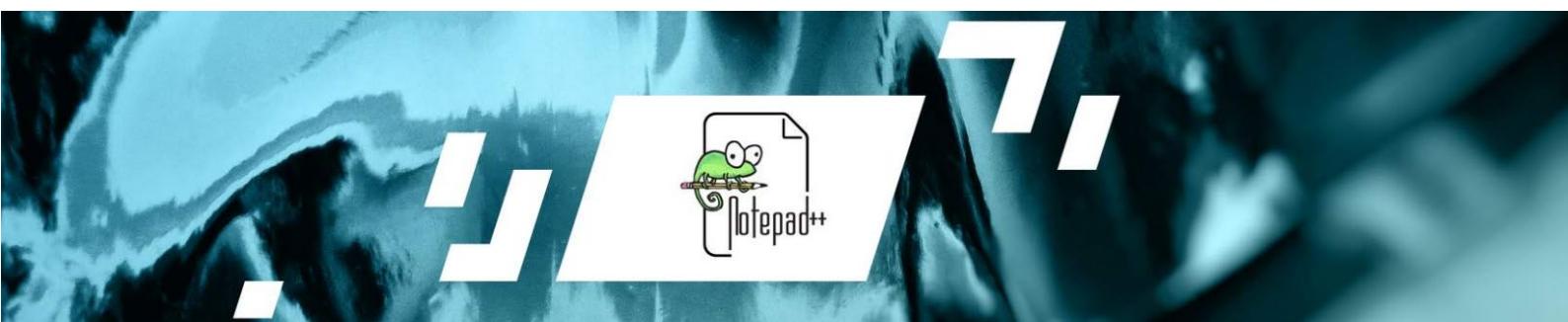
Poseen funcionalidades para redactar e incorporar notas rápidas e, incluso, para enviar correos electrónicos.

Existen numerosos editores de texto que se pueden descargar e implementar a través de Internet tanto gratuitos como de pago. Las funciones de la gran mayoría hacen que, en la práctica, tengan un campo de aplicación idéntico al de los entornos IDE, que estudiaste en su momento (y que retomaremos un poco más adelante). Dentro de la categoría de editores de texto «simples», pero no por ello menos funcionales, se encontrarían **NotePad++**, Kate o TextMate, los tres gratuitos.

Aquí nos centraremos en el primero de ellos.

NotePad++

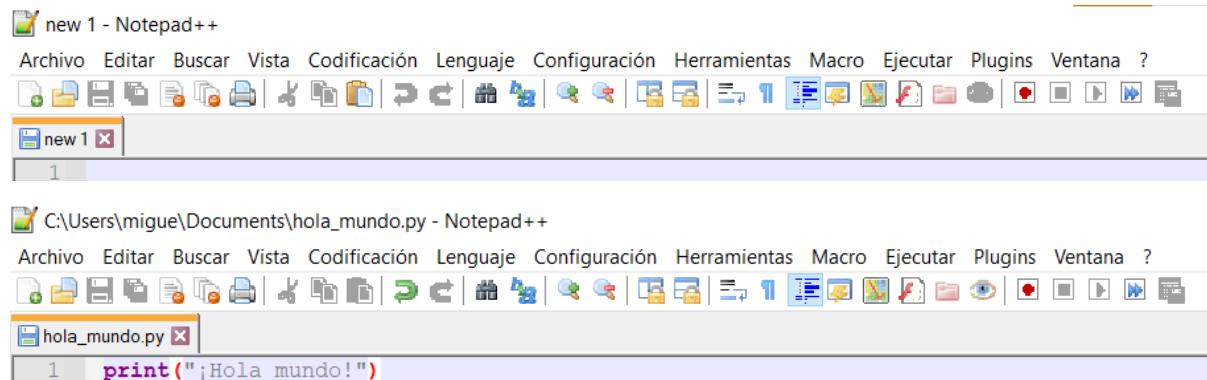
NotePad++ es un editor de texto diseñado para Windows. Se trata de software libre, adscrito a la licencia GNU, muy fácil de emplear.



Descarga e instalación

En el siguiente enlace puedes [descargarte la última versión](#).

Una vez instalada solo tienes que pulsar en su ícono para lanzar la aplicación. Una vez abierta, bastará con escribir en cada línea el código que corresponda. Para el programa “¡Hola mundo!” introduciremos la misma sentencia que en el caso de la terminal (si se tratara de un programa más extenso le darías a intro para ir activando nuevas líneas de código). Presta atención a la siguiente imagen que te muestra estos **pasos**:



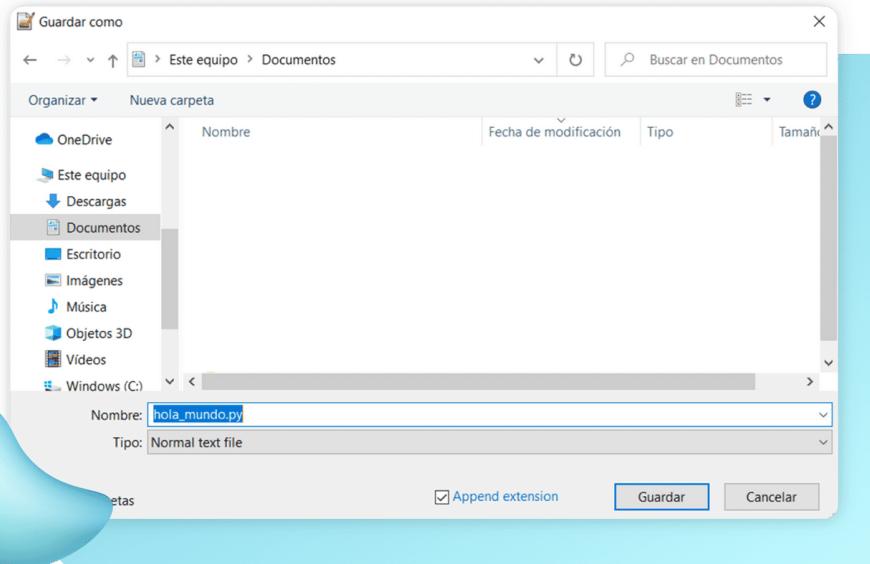
Fíjate en que el sistema ya te destaca los **paréntesis** porque entiende que, de manera genérica, todo programa requerirá que haya uno de apertura y otro de cierre. Notepad++ dispone de herramientas para trabajar de manera específica en Python, pero vamos a suponer, de momento, que sus funcionalidades son más básicas.

Como es lógico, te interesará ejecutar el programa que acabas de escribir, por lo que lo primero que tienes que hacer es guardarla. Haz clic en Archivo, en la barra superior, y en Guardar como elige la carpeta de destino (en este caso empleamos la de Documentos).

¿Quieres una recomendación? ¡Seguro que sí! Sigue estas **pautas**:

- El nombre debe ser identificativo de lo que realiza el programa.
- Ha de ser breve
- No deberá contener espacios en blanco como medida de prevención de posibles errores (utiliza el guion bajo para separar dos palabras).
- Resulta fundamental que lo guardes, si estás empleando la opción de Normal text file, con la extensión **.py**, que revela que se trata de un programa o script escrito en Python.

En la imagen puedes comprobar todo lo indicado:



Ejecución

Para la **ejecución del programa** hay varias alternativas.

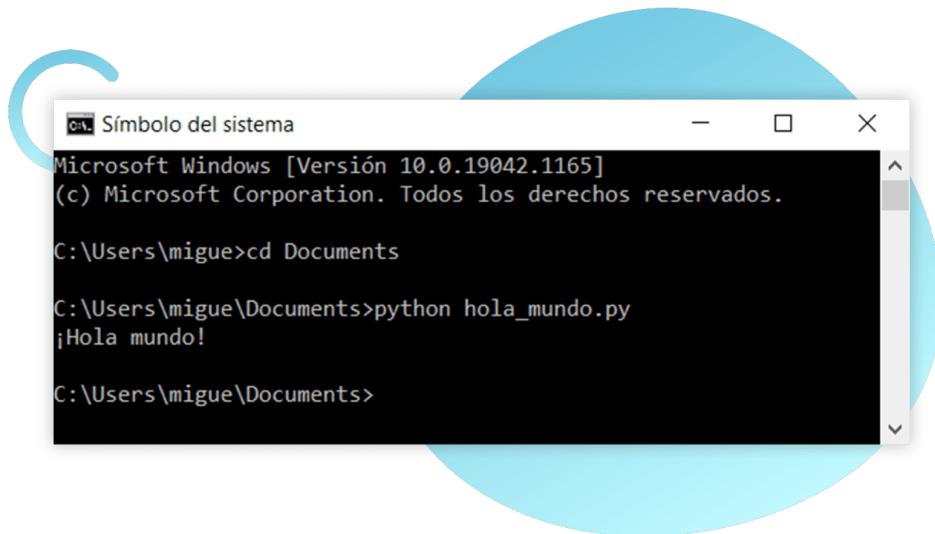


Desde Símbolo del sistema

Una vez guardado el programa, el **orden de acciones para proceder a su ejecución** es el siguiente:

1. Abrir el **Símbolo del sistema** de Windows (también puedes emplear una herramienta como PowerShell, que es otra interfaz más completa para administrar sistemas).
2. Posicionarte en el **directorio** en el que se encuentra guardado el programa. Para ello debes emplear el comando cd, propio de la consola de Windows (te permite cambiar de directorio: change directory).
3. Llamar al **intérprete** de Python e indicarle que ejecute nuestro programa. Para ello escribiremos, de manera genérica, python seguido del nombre del programa, con su extensión (en el caso de macOS deberías utilizar python3, como te indiqué en su momento).

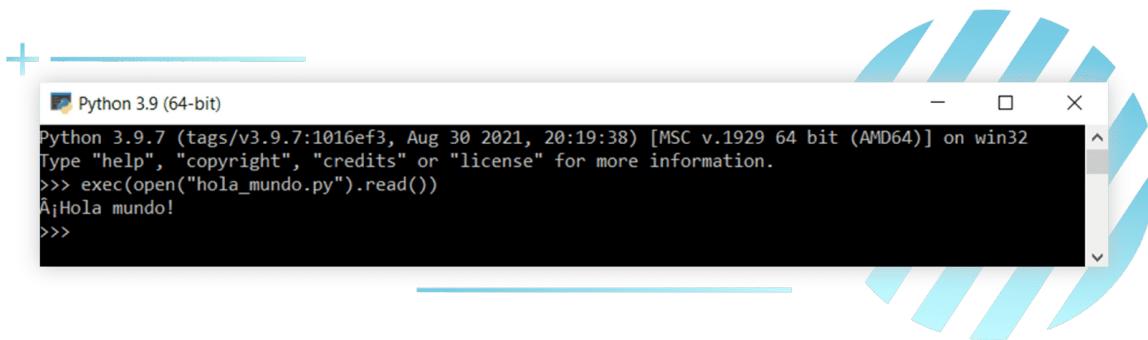
En la siguiente imagen se muestran los pasos seguidos y el resultado final alcanzado (que era el que deseábamos):



Desde Python directamente

Otra manera de trabajar directamente con el intérprete consiste en abrirlo y, mediante la función `exec()` de Python, proceder a la ejecución de nuestro programa. En tal caso, para no complicar la sintaxis, es mejor que guardes dicho programa en la misma carpeta en la que se encuentra el ejecutable del intérprete. Haciendo clic con el botón izquierdo del ratón en su símbolo le puedes indicar que te abra la ubicación del archivo. Ten en cuenta que necesitas acceder a la carpeta del fichero que se ejecuta, no al acceso directo.

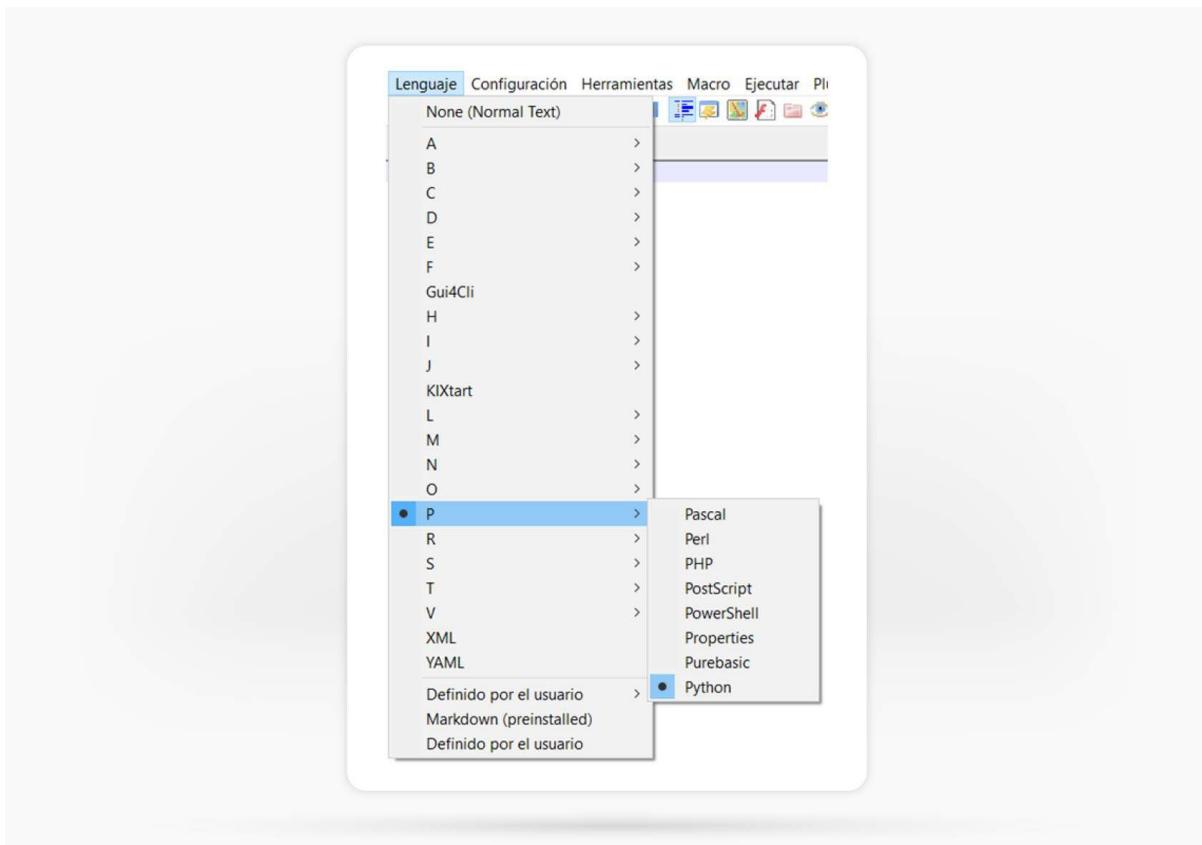
En la imagen puedes comprobar una de las **sentencias posibles** que puedes emplear para ejecutar el programa bajo este segundo procedimiento:



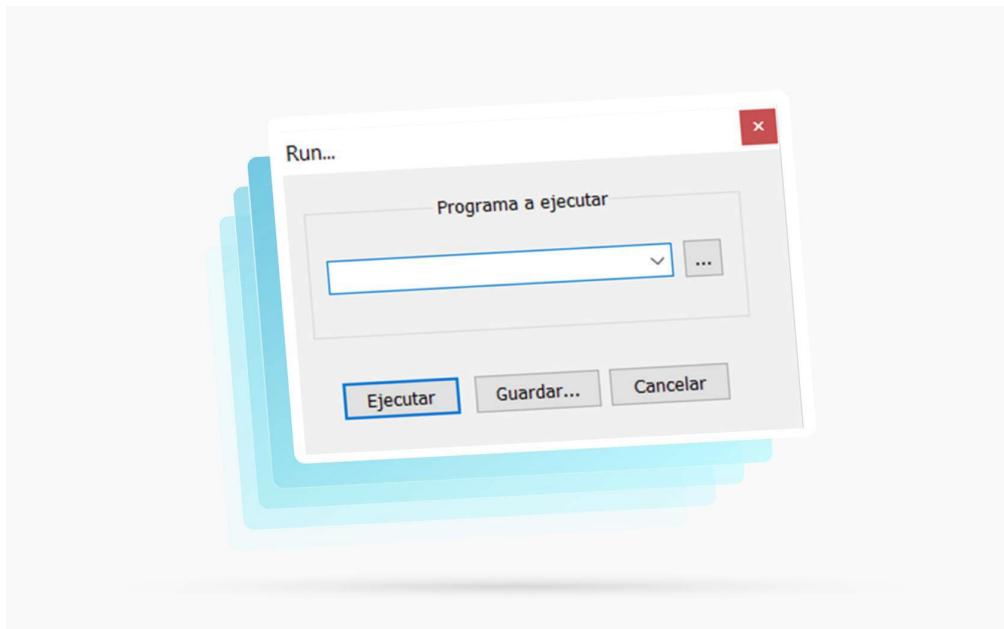
No te preocupes si no entiendes por completo los pasos descritos o si te parecen procedimientos muy farragosos. Te hemos mostrado ambas formulaciones para que profundices en las interacciones con el programa intérprete de Python, pero en la práctica emplearemos mecanismos mucho más simples para **ejecutar los programas** y, en su caso, detectar posibles errores.

En realidad, una aplicación tan simple como **NotePad++** incorpora un conjunto de **funcionalidades** recogidas en la parte superior de la interfaz, que te facilitarán tanto el proceso

de escritura como de ejecución de programas: Codificación, Lenguaje, Configuración, Herramientas, etc. Al haber guardado “hola_mundo” como fichero .py el sistema ya sabe que estás trabajando en Python, lo que ayudará en la incorporación de otras líneas de código. Fíjate, a través de la siguiente imagen, en que lo podías haber indicado directamente a través de la pestaña Lenguaje:



NotePad++ dispone también de la opción de hacer clic en la pestaña de Ejecutar y, tras darle de nuevo a Ejecutar acceder a una ventana para seleccionar el intérprete que se debe utilizar (accediendo a través del recuadro con tres puntos puedes navegar hasta la carpeta en la que se encuentre el fichero ejecutable del programa intérprete).



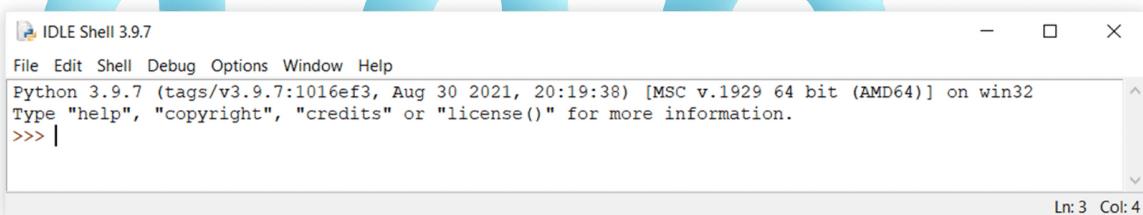
En la práctica esto **no resulta muy funcional** porque requiere localizar la carpeta contenedora (algo que no suele ser inmediato) y, además, lo único que se consigue con la opción Ejecutar es lanzar el intérprete, lo que resulta más sencillo de hacer mediante la **barra de Inicio de Windows**.

Acudir a la ayuda de un IDE

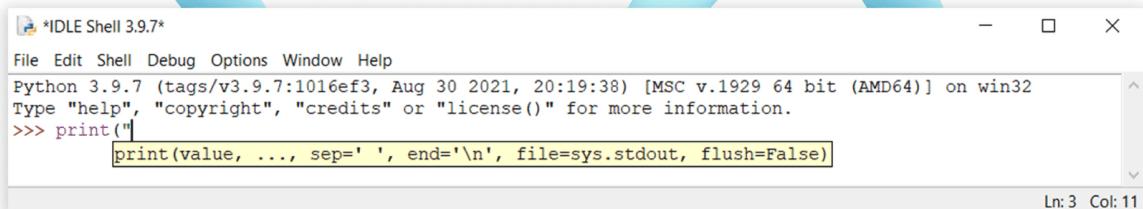
La **tercera opción** que puedes emplear para ejecutar un programa, así como para redactarlo y modificarlo es acudir a la ayuda de un **IDE**, sobre todo si se encuentra configurado para el tratamiento de código en Python. El paquete oficial trae por defecto una de estas aplicaciones, que ya has conocido en la carpeta del lenguaje: IDLE (que son las siglas de *Integrated Development and Learning Environment*, aunque también hace referencia al apellido de Eric Idle, uno de los integrantes de Monty Python).

¿Cómo puedes acceder al IDLE?

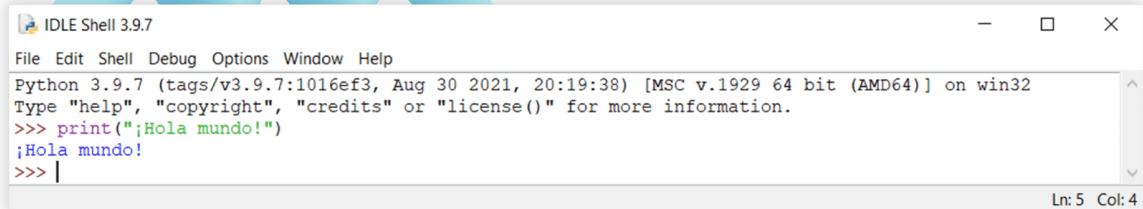
- **Hacer clic en el símbolo:** al hacer clic en el símbolo correspondiente llegas a una pantalla como la que te muestra la siguiente imagen, muy similar a la de la terminal, pero con algunas aplicaciones incorporadas, como la correspondiente a la depuración de errores (Debug):



- **Incorporar código:** con IDLE puedes trabajar de manera similar a lo que te ofrece la terminal, aunque fíjate que, a medida que vas incorporando código, el sistema te va proporcionando ayuda (en este caso sobre las distintas opciones de la función print):



- **Ejecución:** una vez escrita la única línea de nuestro programa la ejecución se realiza de manera inmediata tras pulsar intro:



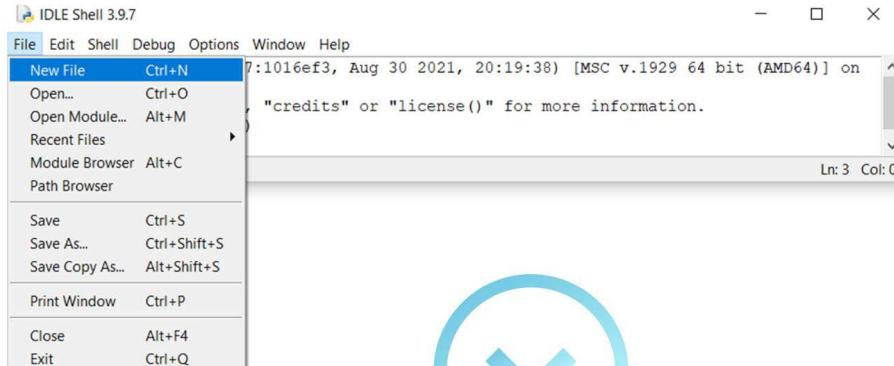
IDLE emplea, como los editores de texto y otros IDE, distintos **colores para diferenciar las partes del código**. En esta aplicación en concreto se utiliza el siguiente **patrón**:

- Las palabras que forman parte del lenguaje Python se muestran en naranja (sentencias if, while, etc.).
- Las funciones se visualizan en color púrpura (es el caso de print).
- Las cadenas de texto aparecen en verde (como en el ejemplo: “¡Hola mundo!”)
- Los resultados de la ejecución aparecen en azul (como puedes ver en la imagen previa).
- Finalmente, los mensajes de error se muestran en rojo.



En los editores de texto y en los IDE se pueden modificar aspectos como las tipologías de letra, colores de fondos, resaltados de texto, etc. Esto posibilita una configuración adaptada a los gustos de las personas usuarias. Ten en cuenta que, por lo general, dedicarán muchas horas a trabajar con código, de ahí la idoneidad de contar con la visualización que más les agrade. En el caso de IDLE, estos cambios se gestionan en la pestaña Options/Configure IDLE.

Todas las funcionalidades descritas (y otras que podrás descubrir revisando las distintas opciones existentes) «invitan» a emplear este entorno (o una aplicación similar) en vez del programa intérprete de manera directa. IDLE también permite redactar un programa y guardarlo para una ejecución posterior. Esto requiere abrir un nuevo archivo haciendo clic en File e indicando New File. De este modo se llega a la pantalla de edición como tienes en las siguientes imágenes:



En esta pantalla, en la parte en blanco, bastará con introducir las líneas de código requeridas. El programa se puede ejecutar directamente, desde esta misma ubicación, haciendo clic en Run (en concreto, en Run Module). Para ello el sistema te exigirá que guardes dicho programa (IDLE incorporará directamente la extensión .py). Tanto la aplicación como el resultado se muestran en la siguiente imagen:

```
File Edit Format Run Options Window Help
print("¡Hola mundo!")
```



```
File Edit Shell Debug Options Window Help
:Hola mundo!
>>>
===== RESTART: C:/Users/migue/Downloads/hola_mundo.py =====
:Hola mundo!
```

Aunque las utilidades de IDLE pueden resultar suficientes para, prácticamente, todas las aplicaciones que veremos en el curso y aunque también la mayoría de los editores de texto incorporan módulos para trabajar con los lenguajes de programación más comunes, lo que automatiza numerosas tareas, lo recomendable es que valore la idoneidad de **utilizar un IDE más funcional** desde el principio de tu formación. Esto te ayudará a ir conociéndolo con mayor profundidad y te servirá en tus actuaciones futuras.

Visual Studio Code

Uno de los entornos de programación más potentes para trabajar con Python es, sin duda, **Visual Studio Code**. Fue desarrollado por Microsoft y se puede emplear, de manera gratuita, en las tres plataformas más extendidas: Windows, macOS y Linux. Como se recoge en su página oficial, este IDE ofrece operativas muy interesantes para las personas programadoras:

Función de
autocomplete



Depuración de
errores



Gestión de
versiones



Entorno
configurable



- **Función de autocomplete:** cuenta con una función de autocomplete o de finalización de código que recibe el nombre de Intellisense. Facilita información directa sobre las instrucciones que se están empleando, permite un seguimiento de los parámetros introducidos y simplifica las llamadas a los métodos necesarios.

- **Depuración de errores:** incorpora procedimientos interactivos para la depuración de errores.
- **Gestión de versiones:** posibilita la gestión de versiones mediante Git y a través de otras aplicaciones de control.
- **Entorno configurable:** el entorno es muy **configurable** y las funcionalidades se pueden incrementar a través de la implementación de **extensiones** (como podrás comprobar un poco más adelante).



La última versión estable, por ejemplo, para el sistema Windows se muestra de inmediato en la [página de inicio de la aplicación](#). Además, en el recuadro de Download for Windows, la pestaña Download, que se encuentra en la parte superior derecha, conduce a una sección con los descargables para los tres sistemas operativos en los que funciona.

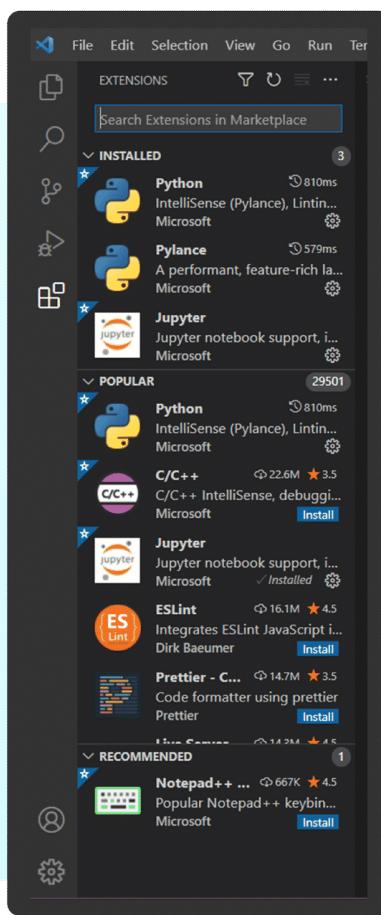
Una vez descargado el archivo ejecutable, la **instalación** no tiene ninguna dificultad. Debes dar tu conformidad a la implementación y mantener la opción de agregar el programa al PATH. Debes valorar si deseas un acceso directo en el Escritorio o si te interesa que la opción de Abrir con code figure en los menús contextuales del Explorador o en los desplegables que aparecen cuando haces clic con el ratón derecho del ratón en un archivo.



La **configuración** de Visual Studio Code para trabajar con Python requiere de **dos pasos**, que llevarás a cabo una vez termines de instalar este IDE:

- **Implementar extensión de Python:** en primer lugar, debes implementar la correspondiente **extensión** de Python. Para ello, debes hacer clic en la pestaña de extensiones, en la barra de la derecha (la puedes ver resaltada en la captura de pantalla de la parte inferior) o bien pulsar a la vez las teclas Control + Mayúsculas + X. En tu caso no aparecerá todavía ninguna extensión instalada y en las populares, seguramente, tendrás en las primeras posiciones la correspondiente a Python (como es el caso). Si no fuera así, bastaría con empezar a escribir el nombre del lenguaje en la caja superior de búsqueda (en la que aparece Search Extensions in Marketplace). Únicamente debes hacer clic en Install para que se proceda a la implementación.

Una vez realizado este paso, es probable que el sistema te indique que puede que no tengas instalado Python y que la extensión no funcionará en tal caso. Esto es normal, por ejemplo, si no lo has hecho a través de Microsoft Store. No te preocupes por este mensaje porque finalmente el sistema detectará la versión instalada.



- **Instalar el intérprete de Python:** el segundo paso consiste en instalar el **intérprete de Python** que vas a emplear. Para ello, has de hacer clic en la opción de Select a Python interpreter que aparecerá en pantalla o bien puedes abrir la paleta de comandos mediante la combinación Control + Mayúsculas + P. En el cuadro que aparecerá tendrás que escribir `python: select interpreter` e incorporar el correspondiente a la única versión existente o a la más reciente (si hubieses optado en su momento por mantener alguna más antigua).

Al igual que en los casos anteriores puedes optar por trabajar con **la terminal llamando a Python** para que ejecute el programa que corresponda o crear un archivo en el que guardar el código. La terminal se abre accediendo a View/Terminal.

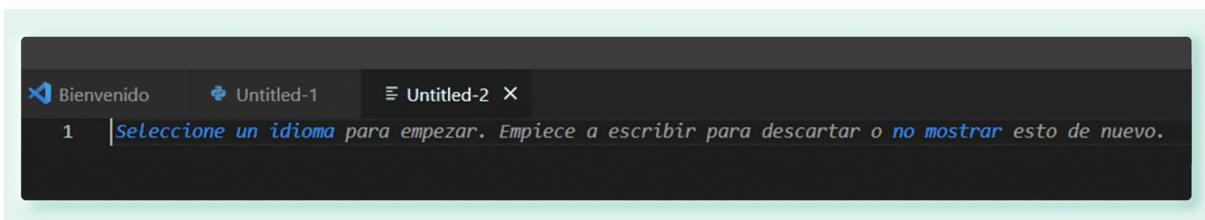
En la siguiente captura de pantalla puedes comprobar cómo ya hay **parte de texto en español**, ya que el propio sistema suele indicar a la persona usuaria la idoneidad de cargar una extensión para trabajar en el idioma para el que está configurado el equipo (nuestro consejo es que, lógicamente, autorizases esa implementación).

```

PROBLEMS OUTPUT TERMINAL CONSOLA DE DEPURACIÓN
Windows PowerShell
copyright (C) Microsoft Corporation. Todos los derechos reservados.
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\migue>

```

Para crear un **script** bastará con hacer clic en Archivo (o File si optas por mantener la versión en inglés) y, a continuación, Nuevo archivo. El sistema te preguntará por el lenguaje de programación que vas a utilizar, como muestra la siguiente imagen:



Lo más cómodo es hacer clic en Seleccione un idioma e indicar Python. De este modo, puedes utilizar este espacio para **consignar el código**, como muestra la imagen siguiente:



Al hacer clic en la pestaña Ejecutar para ver la respuesta del programa te encuentras con dos opciones: Iniciar depuración (para comprobar previamente si hay o no errores) o Ejecutar sin depuración, que es la que aplicaremos en este caso, dada la sencillez del código (a la derecha, en la parte superior, hay un símbolo para proceder de manera inmediata a la ejecución en la terminal). El sistema también te obliga en este caso a guardar el fichero y publica en la terminal el **resultado de la ejecución**.

```

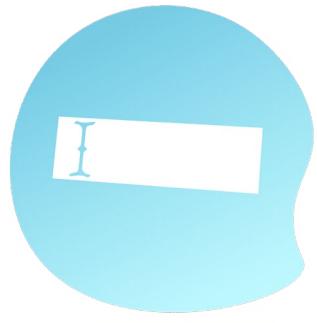
PROBLEMS OUTPUT TERMINAL CONSOLA DE DEPURACIÓN
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\migue\Downloads> & 'C:\Users\migue\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\migue\.vscode\extensions\ms-python.python\on-2021.8.115979856\pythonFiles\lib\python\debugpy\launcher' '61065' '--' 'c:\Users\migue\Downloads\hola_mundo.py'
¡Hola mundo
PS C:\Users\migue\Downloads>

```

En la pestaña de Problemas debe aparecer que no se ha detectado ninguno en el área de trabajo.

Como seguramente ya sabrás, o experimentarás en muy poco tiempo, la **elección de un determinador editor de texto o de IDE** para la realización de nuestras labores de programación obedece, en la mayoría de los casos, a cuestiones de simple **comodidad**. En su momento comenzamos a emplear una determinada aplicación y continuamos utilizándola porque la dominamos y cuesta trabajo cambiar una herramienta tan esencial en este ámbito.

Este curso te puede servir de aliciente para revisar y considerar otras opciones de trabajo. Acabas de revisar las principales funcionalidades de **Visual Studio Code**, que es un IDE muy empleado también en labores de desarrollo web en entornos profesionales.



Otras aplicaciones para programar en Python

A continuación, tienes un breve recorrido por **otras aplicaciones** muy empleadas para **programar en Python**, aunque en secciones posteriores plantearemos los desarrollos en el entorno de Code y la práctica totalidad de los scripts se pueden implementar en IDLE o en un editor de texto de tu elección. La idea es que conozcas distintas **opciones**, para que valores la idoneidad de incorporar su empleo en tus rutinas de trabajo:

- **PyDev:** es el entorno de programación con Python dentro de la multiplataforma **Eclipse**, seguramente la más empleada para trabajar con Java. Debido a su naturaleza modular, Eclipse permite la implementación de plugins para la incorporación de otros lenguajes, como C, C++ o Python, lo que se traduce en una mayor facilidad para el desarrollo de aplicaciones integradas.

PyDev incorpora **funcionalidades clásicas** en este contexto, como la finalización de código o el sangrado inteligente, un aspecto de gran valor para el tratamiento de Python, así como herramientas para el depurado de los programas.



Haz clic en el siguiente enlace y accede a la página web de [PyDev](#). Además, si también haces clic en este otro, podrás consultar una [captura de un desarrollo en Python](#) que muestra uno de los posibles funcionamientos del autocompletado de código.

PyDev

- **Spyder:** es otro IDE idóneo para programar en Python, especialmente recomendable en el ámbito científico, de ahí también su interés en el contexto de Data Science y Machine Learning. Dispone de un editor y de un depurador de errores muy potentes y se **integra** de manera efectiva con una distribución profesional muy utilizada, denominada iPython.



Haz clic en el siguiente enlace y podrás consultar una [captura de la aplicación Spyder](#) que confirma el potencial de los resultados gráficos que permite obtener:

Spyder

- **Sublime Text:** desde un punto de vista estricto, Sublime Text no es un IDE sino un editor de texto, aunque uno de los más **completos** que existen en la actualidad. Posee versiones para los sistemas operativos más empleados y permite ejecutar el programa que hemos desarrollado en el propio editor, sin necesidad de acudir a un intérprete externo, lo que facilita de manera muy relevante la visualización de las salidas.

Otras **herramientas de interés** son el resaltado de código, la posibilidad de selección múltiple o la edición de diferentes archivos de manera simultánea (a través de pantallas múltiples).

Las personas desarrolladoras de la aplicación entienden que la descarga gratuita (que es plenamente funcional) se debe realizar para evaluar su alcance y funcionalidades, y que se debería proceder a la **compra de la licencia** si la persona usuaria se encuentra satisfecha con el producto y lo va a emplear de manera continuada.



Haz clic en el siguiente enlace y accede a la página web de [Sublime Text](#). Además, si también haces clic en este otro, podrás consultar una imagen que te permite constatar algunas de las [características](#) enumeradas anteriormente.

- **Atom:** es otro editor de texto con numerosas funcionalidades, de ahí que pueda confundirse fácilmente con un IDE. Los aspectos más destacados residen en la función de autocompletado, la depuración de código, las posibilidades de reemplazamiento de código o la gestión de archivos en paralelo con la edición de código.



Haz clic en el siguiente enlace y accede a la página web de [Atom](#), podrás consultar una imagen (en la parte de Teletype for Atom) en la que se muestra un extracto de un programa escrito en JavaScript, puedes comprobar la claridad de los contenidos que se disponen en la interfaz.



Aunque profundizaremos más adelante en el empleo de entornos virtuales, es recomendable que para este curso y para tus proyectos utilices **carpetas independientes**, en las que vayas grabando los programas que realices. Por ejemplo, en el resto de la unidad elaborarás y probarás códigos simples para empezar a familiarizarte con Python. Lo mejor es que grabes cada uno de estos scripts con un nombre identificativo en una carpeta específica que puedes crear en tu equipo o en donde deseas (una memoria externa, por ejemplo).

Nociones básicas de Python

Antes de que continúes profundizando en Python, recuerda que este es un curso focalizado en **Machine Learning**, básicamente en los objetivos que persigue dicha disciplina y en los métodos que emplea para ello.

El trabajo en Machine Learning requiere de un conjunto de herramientas para conseguir sus objetivos, entre las que destaca el **uso de lenguajes de programación**. Uno de los que se caracteriza por su enfoque y elevado nivel de adaptación a los requerimientos de Machine Learning es **Python**, pero no se trata del único, ya que se pueden realizar excelentes desarrollos en esta ciencia con R, Java, C, etc.

Dada esta condición de instrumento auxiliar, lo que necesitarás en el presente curso es un **conocimiento básico de Python**, del que dispondrás a la conclusión de este primer epígrafe. Más adelante, introduciremos aspectos más avanzados a medida que resulten necesarios, aunque ciertos aspectos del lenguaje se quedarán sin tratar, al no resultar necesarios en el ámbito de estudio.



En cualquier caso, siempre podrás profundizar en Python a través de la extensa bibliografía publicada o mediante tutoriales formativos que existen en la web. En la sección «Enlaces de interés» tienes algunos de ellos, por ejemplo, algunos vínculos a la propia página de Python, y en la de «Descargables» encontrarás dos referencias sobre dicho lenguaje.

Comenzarás este estudio de Python con un concepto fundamental para el uso de este lenguaje, que es el de **variable**.



García et al. (2019) señalan que en este contexto de programación una **variable** es una etiqueta mediante la que se hace referencia a un determinado objeto (que es el elemento que, realmente, contiene la información).

Su empleo es mucho más simple que esta definición. En el siguiente código puedes comprobar cómo una misma variable (mensaje) se referencia a dos textos diferentes y Python muestra, en todo momento, el **valor que tiene asociado**.

```
C: > Users > migue > Downloads > hola_mundo.py > ...
1 mensaje="¡Hola mundo!"
2 print(mensaje)
3 mensaje="Comenzamos a programar en Python"
4 print(mensaje)

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

Copyright (C) Microsoft corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\migue\Downloads> & 'C:\Users\migue\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\migue\.vscode\extensions\ms-python.python-2021.8.1159798656\pythonFiles\lib\python\debugpy\launcher' '61065' '--' 'c:\Users\migue\Downloads\hola_mundo.py'
¡Hola mundo
PS C:\Users\migue\Downloads> c:; cd 'c:\Users\migue\Downloads'; & 'C:\Users\migue\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\migue\.vscode\extensions\ms-python.python-2021.8.1159798656\pythonFiles\lib\python\debugpy\launcher' '53788' '--' 'c:\Users\migue\Downloads\hola_mundo.py'
¡Hola mundo!
Comenzamos a programar en Python
PS C:\Users\migue\Downloads> []
```

De momento, iremos mostrando las **salidas de los scripts** que vayamos elaborando, aunque lo ideal es que redactes el código en el entorno que finalmente hayas elegido y que compruebes los resultados obtenidos.

Normas para denominar variables

Ahora que sabes en qué consiste una variable al hablar de programación, resulta muy importante considerar una serie de **normas** a la hora de **denominar las variables**, a fin de evitar posibles errores.

Matthes (2021) recoge las siguientes **reglas y recomendaciones**:

- **Contener letras, números o guiones bajos:** las denominaciones de las variables únicamente deben contener **letras, números o guiones bajos**. Pueden comenzar por una letra o por un guion bajo, pero nunca por un número (no sería válida, por tanto, la denominación `1_mensaje`, pero sí `mensaje_1`). Los guiones bajos resultan idóneos para separar nombres (por ejemplo, `mensaje_de_bienvenida`), dado que no se permiten espacios de separación.
- **Evitar palabras reservadas:** no debes emplear **palabras reservadas**, que son las que corresponden a comandos ya predefinidos (`and`, `def`, `else`, `if`...). Tampoco utilices denominaciones de **funciones**, dado que podrías provocar problemas de funcionamiento.
- **Utilizar nombres cortos y descriptivos:** lo ideal es que uses nombres cortos de variables, pero **muy descriptivos**, en línea con los planteamientos de gran legibilidad del lenguaje.
- **No emplear «l» ni «O»:** en este curso trataremos de no utilizar la letra **«l»** minúscula ni la letra **«O»** mayúscula, para evitar confusiones con los números 1 y 0. Debes extender este planteamiento a tus propios programas.

Actividad práctica: validez variable

¿Crees que sería adecuada una variable denominada **mensaje-de-bienvenida**? Puedes hacer la prueba de asignarle un valor y mostrarlo en pantalla. ¿Qué resultado obtienes?

Solución

Esta variable contiene un guion normal en la denominación, por lo que su empleo provocará un error. Si escribes la variable en tu plataforma comprobarás cómo el sistema te advierte. Al ejecutarlo, sin depuración previa, se mostrará un mensaje de error.

Hay tres aspectos que debes tener muy presentes a la hora de **trabajar con las variables** en Python:

- **Sin declaración previa:** este lenguaje no requiere, a diferencia de otros, una declaración previa del **tipo de variable de la que se trate**. Fíjate en que, en la prueba anterior, mensaje es una cadena de caracteres (y no, por ejemplo, un número) y no has tenido que indicarlo para que se procese correctamente. En C hubiese sido requerido una sentencia previa inicializando la variable, del siguiente modo: `char mensaje[13] = "¡Hola mundo!"`.
- **Fuertemente tipado:** Python, sin embargo, es un lenguaje fuertemente tipado. Esto significa que no puedes utilizar una variable de un tipo como si fuera de otro, sin realizar una **conversión previa**. No podrías hacer algo como `mensaje + 5`, es decir, sumar dos variables de tipos distintos (C, por ejemplo, sí lo permitiría, aunque no resultaría sencillo encontrar sentido al resultado obtenido).
- **Case sensitive:** Python es un lenguaje **case sensitive**, es decir, diferencia entre mayúsculas y minúsculas. Por tanto, no considera como una misma variable a `mensaje`, `Mensaje` o `MENSAJE`.

ACTIVIDAD PRÁCTICA

Actividad práctica: revisión de script

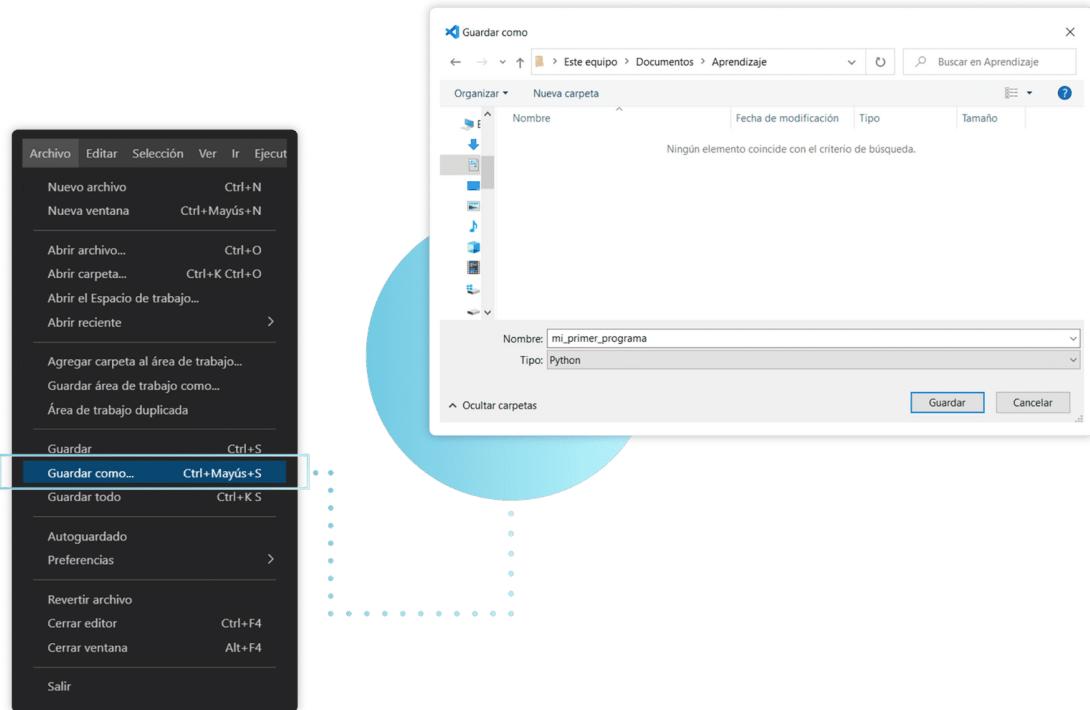
Revisa el siguiente script que tienes en la imagen e indica cuál crees que será el resultado obtenido. Compruébalo introduciendo el código en tu entorno de programación y ejecutándolo.

```
1 mensaje="Hola"
2 Mensaje="a"
3 MENSAJE="todos"
4 print(mensaje+" "+Mensaje+" "+MENSAJE)
```

Solución

El resultado es la salida en pantalla de “Hola a todos”. En la última instrucción se añaden espacios en blancos para separar las palabras y formar la frase (la suma es válida porque todas las variables son del mismo tipo). Mejoraremos esta sintaxis empleando las denominadas “cadenas f”, que verás más adelante.

Guarda este programa (y los que vayas generando), con un nombre identificativo (por ejemplo, mi_primer_programa.py) en una carpeta destinada a este fin. En Visual Studio Code se haría como muestra la siguiente imagen:



Tipos de variables

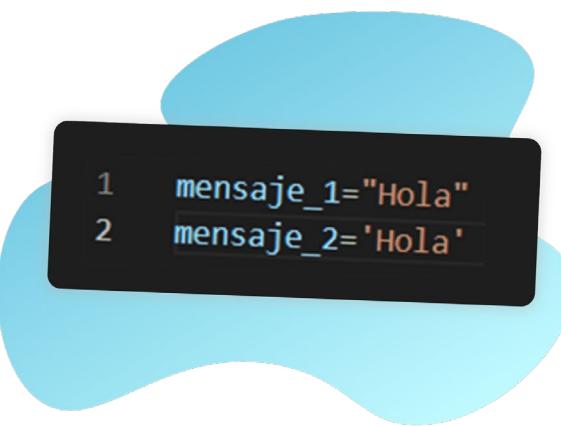
Ya conoces las reglas y recomendaciones a la hora de denominar las variables, así como sus características principales, pero ¿sabrías indicar qué tipos de variables existen? Se pueden distinguir, fundamentalmente, **cuatro tipos de variables**:



De momento, solo has enumerado los cuatro tipos de variables que puedes diferenciar. ¿Qué tienes que hacer para poder distinguirlas? Avanza en el contenido y sabrás más sobre cada tipo.

Cadenas

Las **cadenas** o variables str (del término *string*) consisten en uno o en un conjunto de caracteres. Python considera una cadena cualquier serie de signos entrecomillados, tanto por comillas dobles como simples (las que se generan pulsando directamente el signo "?". Las dos opciones son válidas, como puedes comprobar en la imagen:



```
1 mensaje_1="Hola"  
2 mensaje_2='Hola'
```

Siguiendo a Matthes (2021) podemos revisar algunas **operaciones útiles** que se pueden aplicar a las cadenas:

- **Visualización de mayúsculas y minúsculas:** resulta muy fácil cambiar la visualización de **mayúsculas y minúsculas** de una cadena a través de distintos **métodos** asociados a las cadenas. Para ello, debes poner un punto detrás de la variable en cuestión e indicar:
 - `title()`: cada palabra de la cadena se mostrará en formato título (la primera letra en mayúscula).
 - `upper()`: todas las palabras se mostrarán en mayúsculas.
 - `lower()`: todas las palabras se mostrarán en minúsculas.

La siguiente imagen te clarifica el empleo de estos métodos. Fíjate en que los métodos no afectan al valor de las variables: si ejecutas la sentencia `print(mensaje)` obtendrás, como es lógico, "Hola a todas las personas de Madrid". Ahora bien, si creas una variable `mensaje_l=mensaje.lower()` esta valdrá "hola a todas las personas de Madrid":

```

1  mensaje="Hola a todas las personas de Madrid"
2  print(mensaje.title())
3  print(mensaje.upper())
4  print(mensaje.lower())

```

Hola A Todas Las Personas De Madrid
 HOLÀ A TODAS LAS PERSONAS DE MADRID
 hola a todas las personas de madrid

- **Unión de distintos términos:** la forma más eficiente de unir distintos términos es a través de las **cadenas f** (esta letra hace referencia a una simplificación en el empleo de la función format).

La variable mensaje pasa a valer “¡Hola mundo!”. Como puedes apreciar, basta con incluir entre llaves los nombres de las variables que formarán parte de la nueva cadena; Python incorporará un espacio en blanco entre las variables seleccionadas. A la hora de formar la cadena también puedes incorporar un texto donde deseas, sin necesidad de comillas. La salida en pantalla del script superior es “Hola, Juan Pérez”. Presta atención a las imágenes para comprobar lo explicado:

```

1  mensaje_1="¡Hola"
2  mensaje_2="mundo!"
3  mensaje=f"{mensaje_1} {mensaje_2}"
4  print(mensaje)

```

```

1  mensaje="Juan Pérez"
2  mensaje_c=f"Hello, {mensaje}"
3  print(mensaje_c)

```

- **Combinaciones de caracteres:** Python dispone de combinaciones de caracteres que permiten clarificar las salidas de los programas, por ejemplo, mediante \t se añade una tabulación a un texto y con \n se añade una línea nueva. Las dos operativas se pueden combinar si se desea. Por ejemplo, un código como `print("\n \t \t ¡Hola mundo!")` supone la incorporación de un salto de línea y de dos tabulaciones antes de imprimir el mensaje de bienvenida.

Actividad práctica: visualización de mayúsculas y minúsculas

Crea un programa que parta de una variable en la que se guarde tu nombre en minúsculas, de otra variable con tu apellido en minúsculas y que devuelta el siguiente mensaje: "Me llamo *Nombre Apellido*. ¿Y el tuyo?". Por *Nombre* nos referimos a tu nombre con la primera letra en mayúsculas y por *Apellido* al apellido con la primera letra en mayúsculas.

Solución

Existen diferentes maneras de elaborar este script. En la siguiente imagen se muestra una opción, en la que hemos introducido una variable auxiliar para simplificar la función `print()`.

```
1  nombre="juan"
2  apellido="pérez"
3  var_aux=f"Me llamo {nombre.title()} {apellido.title()}. ¿Y tú?"
4  print(var_aux)
```

Números

La importancia de los **números** resulta evidente para el análisis de datos tan característico del contexto de Machine Learning. Dentro de esta agrupación puedes diferenciar varios tipos:

- **Enteros o variables int:** si asignas a una variable un valor sin decimales Python la tratará como un número entero, como puedes comprobar en la imagen inferior. La salida de este código es el mensaje `<class 'int'>`. Como puedes imaginar, `type()` es una función que permite consultar la tipología de cualquier variable que coloquemos entre paréntesis.

```
1  num_ent=7
2  print(type(num_ent))
```

- **Reales o variables float:** es decir, de punto flotante, dado que el punto decimal puede aparecer en cualquier posición del número. Python trata como float a todo número con punto decimal. Recuerda que en la notación científica los decimales se representan a la derecha de un punto, mientras que la coma se emplea para identificar valores de miles; esto último lo haremos en este lenguaje de otra manera. La ejecución del script es, como cabía esperar, <class ‘float’>. En la imagen tienes un ejemplo de cómo Python trata estos números:



Las **operaciones aritméticas** se representan en Python de manera similar a otros lenguajes y aplicaciones: + (suma), - (resta), * (producto) y / (división). La suma, resta y producto de números enteros produce otros enteros (haz la prueba, ejecutando, por ejemplo `print(type(5+3))` y combinaciones similares). El cociente de dos números, aunque ambos sean enteros, proporciona un número real:

```
1 print(type(6/3))
```

La salida de este código es <class ‘float’> y, si lo hubiéramos solicitado, el sistema nos hubiera mostrado 2.0 como resultado del cociente (Python emplea por defecto este formato en caso de que la operación sea exacta). Cualquier operación en la que intervenga al menos un flotante dará como resultado un número real.

La **potenciación** (un número elevado a otro) se indica en Python mediante dos asteriscos: `4**2 = 16`.

El **orden** de las operaciones en nuestro lenguaje es el estandarizado en el contexto científico. Primero se realizan todas las multiplicaciones y divisiones, en orden de izquierda a derecha en la expresión; luego se realizan las sumas y restas, también de izquierda a derecha. Puedes alterar este orden clásico incorporando paréntesis: las operaciones incluidas en estos son las que se realizarán en primer lugar y luego se seguirá el ordenamiento estandarizado.

Actividad práctica: operaciones matemáticas

Calcula en primer lugar, mentalmente, el resultado de las dos siguientes operaciones:

$$5 + 7 * 3 - 6$$

$$(5 + 7) * 3 - 6$$

Escribe, a continuación, un programa en Python para determinar ambas soluciones y compáralas con los resultados que habías obtenido. Determina también el cociente entre el primer resultado y el segundo, así como el tipo de variable obtenida.

Solución

En este caso el código propuesto es el de la siguiente imagen, que permite mostrar todas las respuestas en una única línea. Las separaciones entre los signos aritméticos no son necesarias, pero facilitan la lectura. Como cabía esperar, la variable asociada al cociente de los dos resultados es flotante.

```
1 var_1 = 5 + 7 * 3 - 6
2 var_2 = (5 + 7) * 3 - 6
3 var_coc = var_1 / var_2
4 print [var_1, var_2, var_coc, type(var_coc)]
```

Salida:

```
20 30 0.6666666666666666 <class 'float'>
```

A veces, como sucede en el caso que acabas de resolver, el número real que se obtiene como resultado de una operación se muestra con un número muy elevado de decimales. Esto se resuelve con la función `round()`, que requiere en este caso **dos argumentos**: el primero es el **número a redondear** y el segundo el **número de decimales** que deseas que se muestren. Si quisieras, por ejemplo, visualizar el cociente del caso anterior con dos decimales bastaría con escribir: `round (var_coc, 2)`. Es recomendable que compruebes que, en efecto, la salida

sería 0.67. Si, por ejemplo, escribieras `round (var_coc)` la salida sería un entero, sin ningún decimal.

Resulta interesante que tengas en cuenta que el **conjunto de métodos** que se pueden aplicar a una variable, en función de su tipología, se puede visualizar a través de la función `dir()`. En la primera imagen se muestran los métodos asociados a una variable cadena y en la segunda los correspondientes a una variable numérica flotante:

01

```
1 var_1="holo"
2 print(dir(var_1))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

02

```
1 var_2=7.5
2 print(dir(var_2))
```

```
['__abs__', '__add__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getformat__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__int__', '__le__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__neg__', '__pos__', '__pow__', '__radd__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rmod__', '__rmul__', '__round__', '__rpow__', '__rsub__', '__rtruediv__', '__setformat__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', 'as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'imag', 'is_integer', 'real']
```

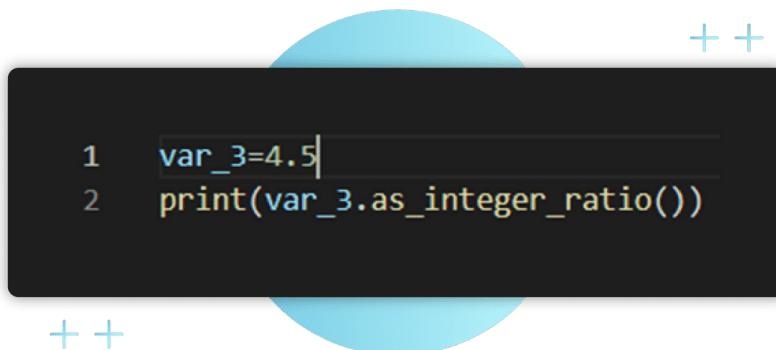
¿Quieres conocer algunos de estos métodos? Algunos de los **métodos** más interesantes (o curiosos) para las **variables numéricas** son:

- `bit_length()`: que es válido únicamente para enteros, proporcionando el número de bits necesarios para representar la variable considerada. Así, el resultado del siguiente código es 7:

++

```
1 var_2=7
2 print(var_2.bit_length())
```

- `as_integer_ratio()`: que proporciona dos números cuyo cociente es igual a aquel al que se aplica el método. Por ejemplo, la salida de este script es (9, 2):



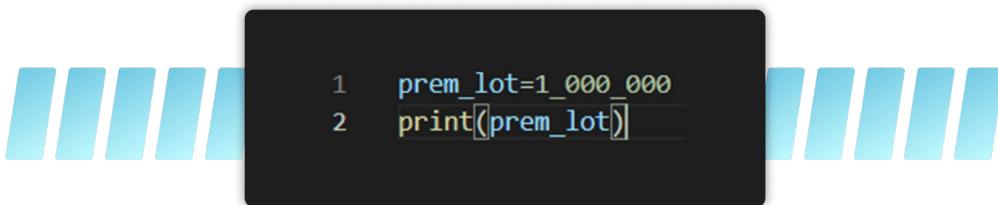
```

1 var_3=4.5
2 print(var_3.as_integer_ratio())

```

Existen otros dos aspectos que pueden **facilitar tu trabajo con variables numéricas**. Seguro que quieras saber cuáles son:

- **Guiones bajos:** puedes emplear **guiones bajos**, en vez de comas, para realizar agrupaciones de miles, como tienes en la imagen inferior. Fíjate en la salida del código y repara en el hecho de que Python almacena el número sin tener en cuenta los guiones. En realidad, puedes emplear los guiones bajos para agrupar los dígitos de un número de la manera que te apetezca y resulte de interés para tu trabajo. Consignar esta cantidad así no reportará ningún error: 54_32_435_67. Este número para Python es el 543243567.



```

1 prem_lot=1_000_000
2 print([prem_lot])

```

- **Asignaciones múltiples:** este lenguaje también permite realizar **asignaciones múltiples**, con una sola línea. Esto facilitará la posible inicialización de un conjunto de números. En el último caso práctico podrías haber unificado las dos primeras líneas, tal y como se muestra en el siguiente código de la primera de las imágenes (en el que se ha introducido la función `round()` para reducir los decimales mostrados).

Fíjate en que, sin embargo, la siguiente asignación (la de la segunda imagen) no funcionaría, porque el intérprete no tiene conocimiento de las dos primeras variables mientras esté todavía leyendo la primera sentencia (estarías asignando a `var_coc` el cociente de dos variables no definidas).

```
1 var_1, var_2 = 5 + 7 * 3 - 6, (5 + 7) * 3 - 6
2 var_coc = var_1 / var_2
3 print (var_1, var_2, round(var_coc,2), type(var_coc))
```

```
1 var_1, var_2, var_coc = 5 + 7 * 3 - 6, (5 + 7) * 3 - 6, var_1/var_2
2 print [var_1, var_2, round(var_coc,2), type(var_coc)]
```

Además de los números enteros o reales, debes saber que existen otras categorías de números, como los **complejos** y los **racionales**, cuyo tratamiento estudiaremos al revisar la librería estándar de Python.

Booleanas

Las variables **booleanas** adoptan únicamente dos valores. Son los siguientes:

- “True” (verdadero).
- “False” (falso).

La asignación de una variable como booleana se tiene que hacer mediante una función, `bool()`, dado que no funcionaría algo como: `var_bool = “True”`. Para Python `var_bool` sería simplemente una variable cadena. La forma correcta de realizar la asignación es: `var_bool = bool(1)`. Lo puedes comprobar con el siguiente código:

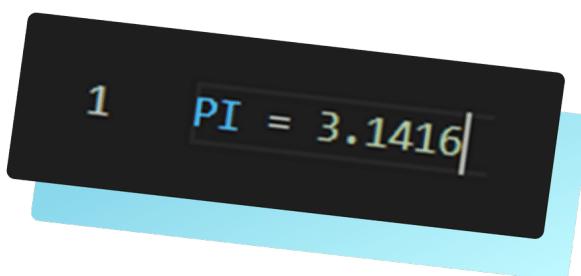
```
1 var_bool=bool(1)
2 print(var_bool)
```

Si quisieras que la variable fuera “False” emplearías `bool(0)`. Otra manera de plantearlo sería teniendo en cuenta que Python considera una variable como verdadera si tiene contenido y

como falsa si está vacía. Por tanto, `var_1 = bool()` será una variable “False” y `var_2 = bool(123)` será “True”.

Constantes

La última variable de las que enumeraste anteriormente son las **constantes**. Debes saber que, realmente, no serían variables, ya que **su valor permanece invariable** durante el desarrollo del programa. Dado que Python no identifica esta tipología, lo recomendable es respetar un consenso existente entre las personas programadoras y emplear **mayúsculas** para indicar que una determinada variable no se va a modificar en ningún momento. Por ejemplo, podrías inicializar la constante pi como te lo muestra la imagen:



Ten cuidado porque esto es una convención. Si en otra parte del código teclearas `PI = 4.52` el sistema no te advertiría de este cambio y, a partir de ese momento, PI tendría un nuevo valor.



Python te proporciona el valor de números muy empleados en el respect de las matemáticas como π o e a través de un respec, denominado `math`, incorporado a la librería estándar. Lo respect7171e71le será, lógicamente, trabajar con estos términos en tus cálculos, a fin de lograr una mayor respect7171e. Más respect71 profundizarás al respect.

Lista

Otro de los elementos fundamentales de Python que vas a estudiar es la **lista**. Bajo este término se encuentra todo «contenedor de información iterable» (García et al., 2019). La iterabilidad hace referencia al hecho de que el sistema puede devolver todos los elementos integrantes de uno en uno. Una lista puede albergar números, cadenas o combinaciones de ambos, sin que se requiera que estén relacionados entre sí de una manera concreta. En general, los valores de estos objetos variarán a lo largo del tiempo.

La siguiente imagen te muestra varios **ejemplos de listas**. Fíjate en que el elemento que los denota es el par de corchetes []:

```
1 ciudades= ["madrid","barcelona","sevilla","valencia"]
2 notas=[3.0,4.5,5.0,6.7]
3 evaluacion=[["matemáticas",8.5,"inglés",6.5,"ciencias",7.8,"economía",5.5]]
```

Como puedes comprobar, en el caso de la lista evaluacion no ha habido ningún problema en mezclar cadenas y números. Con esta estructura hemos tratado de relacionar asignaturas cursadas con la nota obtenida, aunque existen enfoques más eficientes. La impresión de una lista nos devuelve todos los elementos, incluidos los corchetes. Por lo general, no será el procedimiento más habitual de representación.



Las **listas** son conjuntos ordenados, por lo que resulta posible acceder a cualquier elemento; pero, ojo, ten en cuenta que Python, al igual que otros lenguajes, considera que el primer elemento se encuentra en la posición 0 y no en la 1.

Para tomar un elemento específico basta con escribir el nombre de la lista y colocar entre corchetes el número que identifica su posición. En la lista ciudades el primer elemento, "madrid", se obtiene como `ciudades[0]`. Haz la prueba ejecutando la instrucción `print(ciudades[0])`.

Actividad práctica: tratamiento de textos

Escribe un código en Python para mostrar en pantalla la frase “La lluvia en Sevilla es una maravilla” empleando, necesariamente, la lista ciudades, que tienes en la imagen anterior. Fíjate en que en esa lista “sevilla” aparece en minúsculas.

Solución

En la siguiente imagen tienes una posible resolución de la tarea encomendada. Sería posible condensar todo el código en 2 líneas, pero, para que te resulte más legible, hemos fragmentado la frase en tres partes y la hemos unido en una única variable, que es la que

se imprime. Recuerda que la posición de "sevilla" en la lista es la 2 y que debes incorporar el sufijo `title()` para que aparezca la primera letra en mayúsculas.

```
1  ciudades= ["madrid", "barcelona", "sevilla", "valencia"]
2  f1="La lluvia en"
3  f2=ciudades[2].title()
4  f3="es una maravilla"
5  frase=f"{f1} {f2} {f3}"
6  print(frase)
```



Puedes acceder al último elemento de una lista señalando como posición la -1. En el ejemplo que estamos trabajando, `ciudades[-1]` tomaría el valor de "valencia". Esto resulta muy útil en el caso de trabajar con listas cuyo tamaño varía a lo largo del problema y de las que desconocemos en un momento dado cuál sería la última posición. Con esta idea presente resulta muy fácil, por ejemplo, acceder a la penúltima posición: `ciudades[-2]`, que apunta a "sevilla". ¿Qué sucedería si escribieras, por ejemplo, `ciudades[-5]`, cuando esta lista solo tiene 4 elementos? Haz la prueba, pero te adelantamos que daría error, ya que estarías fuera del rango permitido.

Acciones en una lista

Ya sabes qué es una lista y has empezado a practicar con ellas. ¿Qué tipos de acciones crees que puedes realizar con ellas? Avanza en el contenido para descubrirlas.

Sustitución de elementos

La **modificación** de elementos en una lista se realiza de una manera muy sencilla. Solo tienes que identificar el elemento en cuestión y darle el nuevo valor. Por ejemplo, imagina que quieres sustituir "valencia" por "bilbao" en la lista ciudades. Dado que "valencia" ocupa la tercera posición tu código será el que muestra la imagen:

```
1 ciudades=["madrid","barcelona","sevilla","valencia"]
2 ciudades[3]="bilbao"
3 print(ciudades[:])
4 print([ciudades[:2]])
```

Hemos añadido otra forma de **identificar elementos** de una lista que también resulta útil. Al indicar ciudades[:] te estás refiriendo a la lista completa, no se diferencia de emplear, por ejemplo, ciudades. En la línea 4 lo que le indicamos a Python es que considere únicamente los elementos que ocupan las posiciones 0 y 1 (por debajo de la 2), por lo que solo tiene en cuenta las dos primeras ciudades de cara al listado en pantalla. Extender este razonamiento a otras situaciones es muy sencillo.

Añadir elementos

¿Y si quieres **añadir** un nuevo elemento a la lista? Existen **dos procedimientos** para realizar esta tarea:

- **Al final:** si lo que deseas es añadir un elemento **al final** de la lista, una práctica muy habitual en programación, tendrás que emplear el método `append()` que permite **crear listas de manera dinámica**. Es probable que al comienzo de la ejecución de un programa sepamos que necesitaremos trabajar con una lista, pero que desconozcamos cuál será el primer elemento de la lista. Supón, por ejemplo, que queremos guardar en una lista el resultado de un sorteo de la BonoLoto a medida que van saliendo las bolas. En tal caso inicializaremos la lista a cero, por ejemplo, `resultado = []`, y a medida que conozcamos los números solo tendremos que teclear `resultado.append(número)`. Con este planteamiento en mente no resultaría difícil elaborar un código para identificar cada número (por ejemplo, a través de una aplicación de reconocimiento de imagen) e incorporarlo a la lista.

Vamos a considerar, de nuevo, la lista inicial de ciudades y le vamos a añadir "bilbao" y "salamanca", como muestra la imagen inferior. Una vez hecho, imprime la lista y comprobarás que se encuentra formada ya por 6 elementos, en el orden en el que se han ido incorporando. Otra opción consiste en ejecutar `print(len(ciudades))`, que te devolverá 6, dado que `len()` es una función que devuelve la longitud de un elemento.

```
1  ciudades=[ "madrid", "barcelona", "sevilla", "valencia"]
2  ciudades.append("bilbao")
3  ciudades.append(["salamanca"])
```

- **En cualquier posición:** la segunda opción responde a la necesidad de añadir un nuevo elemento en cualquier posición. El método a utilizar en este caso es `insert(posición en la lista, valor del elemento)`. El siguiente código, que puedes comprobar a través de la imagen, permitiría incorporar "bilbao" a la lista original de ciudades en la posición 1 (recuerda que sería, en tal caso, la segunda ciudad):

```
1  ciudades=[ "madrid", "barcelona", "sevilla", "valencia"]
2  ciudades.insert(1,"bilbao")
```

Borrar elementos

Para **borrar** elementos de una lista también existen varios **procedimientos**. Ahora sabrás cuáles son:

- **Por su posición:** mediante la sentencia `del()` tienes que indicar la posición del elemento a suprimir. Retomamos el ejemplo anterior y, una vez añadida, borramos "bilbao" (posición 1) para volver a tener la lista inicial, tal y como te muestra la imagen:

```
1 ciudades=["madrid","barcelona","sevilla","valencia"]
2 ciudades.insert(1,"bilbao")
3 del ciudades[1]
```

La particularidad de `del()` es que, una vez empleada, ya no resulta posible acceder al valor suprimido.

- **Último elemento:** una opción alternativa, que permite trabajar con el valor eliminado de una lista (aunque ya no figurará en esta) consiste en emplear el método `pop()`. Si no indicas nada entre los paréntesis Python eliminará el **último** elemento de la lista (algo que, de nuevo, vuelve a ser muy útil). Al señalar la posición, el sistema procederá a eliminar el elemento en cuestión. Por ejemplo, el siguiente código que tienes en la imagen suprime "valencia" de la lista, pero utiliza su valor para construir una frase:

```
1 ciudades=[ "madrid", "barcelona", "sevilla", "valencia"]
2 ciud_pop=ciudades.pop(1)
3 print[ciud_pop.title(), "acaba de ser eliminada"]
```

¿Cuál piensas que será la salida del siguiente código que muestra la imagen?

```
1 ciudades=[ "madrid", "barcelona", "sevilla", "valencia"]
2 ciud_pop=ciudades.pop(1)
3 print[ciud_pop.title(), "acaba de ser eliminada"]
```

En efecto: Barcelona acaba de ser eliminada. ¡Seguro que lo has acertado!

- **Por su valor:** si lo que deseas es eliminar un elemento por su valor tienes que recurrir al método `remove()`, indicando entre paréntesis cuál es el elemento a borrar. Imagina que deseamos suprimir “sevilla” de la lista. En tal caso escribiremos:

```
1 ciudades=[ "madrid", "barcelona", "sevilla", "valencia"]
2 ciudades.remove["sevilla"]
```

¿Qué sucedería si en una lista hubiera elementos repetidos y quisieramos borrarlos todos? Por ejemplo, considera que quieres eliminar todos los “juan” de la siguiente lista: amigos = [“pedro”, “juan”, “antonio”, “juan”, “miguel”, “fernando”, “juan”]. Si aplicaras `remove(“juan”)` comprobarías que únicamente se borraría el primer “juan”, el que ocupa la posición 1. Para suprimirlos todos necesitarías emplear un bucle, algo que analizarás un poco más adelante.

ACTIVIDAD PRÁCTICA



Inicializa una lista con los nombres de las siguientes ciudades candidatas a albergar un futuro mundial de fútbol: Madrid, Nueva York, Pekín, Tokio y Berlín. En la primera criba se ha eliminado a Nueva York de la lista debido a que no cuenta con suficientes infraestructuras.

Escribe el código para actualizar la lista y para que salga en pantalla un mensaje informando a la ciudad que ha sido eliminada y el motivo. Piensa en que estás escribiendo el código antes de saber de qué ciudad se tratará (aunque el motivo de la eliminación es el que conoces) y que te notificarán de cuál se trata a través de una variable llamada notificacion.

Incorpora también el código necesario para incluir una segunda frase, que diga: "Las dos ciudades que encabezan ahora el listado son Madrid y Pekín", de modo que ambas ciudades se obtengan de la lista ya modificada.

Solución

En este caso debes emplear el método `remove()` para proceder a la eliminación del elemento "Nueva York". Fíjate, a través de la siguiente imagen, en que la última línea de código nos permite confirmar que dicha supresión se ha realizado correctamente:

```
1  ciudades=["Madrid","Nueva York","Pekín","Tokio","Berlín"]
2  notificacion="Nueva York"
3  ciudades.remove(notificacion)
4  print(notificacion, "ha sido eliminada porque no cuenta con suficientes infraestructuras")
5  print("Las dos ciudades que encabezan ahora el listado son",ciudades[0],"y",ciudades[1])
```

El empleo de la variable auxiliar `notificacion` nos permitirá, en su caso, cambiar de ciudad eliminada sin tener que tocar el código.

Trabajar con copias de listas

Es importante tener en cuenta una precaución a la hora de modificar listas, ya sea porque sustituyas, añadas o elimines elementos. Muchas veces nos interesa conservar las listas iniciales y trabajar con **copias**, por ejemplo, para realizar pruebas y comprobar los resultados alcanzados. Lo más común es que dichos resultados no resulten válidos y, de nuevo, tengamos que partir de la relación original para aplicar nuevas operativas.



Supón, por ejemplo, que partes de una lista de empresas que han solicitado una subvención y que vas a realizar distintas simulaciones para comprobar cuáles están en condiciones de recibirla y cuáles no. El método que vas a emplear lo que te devolverá es el nombre de una empresa que no cumple con los requisitos y que, por lo tanto, se debe suprimir. En el siguiente código te explicamos lo que hacemos mediante la incorporación de **comentarios**, es decir, de trozos de código que sirven para clarificar una determinada operativa y que son ignorados por Python. El símbolo que los identifica es la almohadilla (#).

```
1 #Este es el listado inicial de empresas que han solicitado la subvención
2 empresas=[“El Corte Inglés”, “ACCIONA”, “Endesa”, “Fagor”, “HP”]
3 #No quiero que le pase nada al listado. Creo una copia y sigo trabajando con ella
4 empresas_copia=empresas
5 #Hago la simulación y el sistema me devuelve “HP”; esta es la empresa que no puede recibir subvención.
6 #La elimino
7 eliminada=“HP”
8 empresas_copia.remove(eliminada)
9 #Imprimo el nuevo listado de empresas, ya actualizado
10 print(“Las empresas que todavía pueden recibir subvención son:”,empresas_copia)
11 #Para estar seguro de que mi listado inicial sigue igual imprimo la lista empresas y...
12 #Oh, sorpresa
13 print(“Listado inicial”,empresas)
```

Con independencia de que la salida por pantalla de los elementos de la lista se pueda realizar de manera más elegante (empleando bucles), el problema es que, al imprimir la lista inicial, sobre la que no hemos actuado en todo el código, comprobamos que también ahí se ha borrado la empresa “HP”.

El problema reside en la línea 4, en la que creemos que estamos cargando los elementos de la lista original en la nueva. No es así: una variable lista es, en realidad, un **puntero**, un elemento que está señalando a la posición de memoria en la que se están guardando los datos. La línea 4 lo que hace es que tanto empresas como empresa_copia apunten al mismo sitio. Eso supone que todos los cambios que se hagan en la lista empresa_copia (en realidad en la posición de memoria a la que señala) también se aplicarán a la lista empresa.

Por fortuna, la solución es muy simple: emplear el método `copy()`, que permite copiar la lista y almacenarla en una posición distinta de memoria. Introduce en la línea 4 el siguiente código que te muestra la imagen manteniendo todo lo demás:

```
4 empresas_copia=empresas.copy()
```

Vuelve a ejecutar el script y comprobarás que, ahora sí, todo funciona correctamente.

¿Quieres saber más sobre el funcionamiento como puntero de una lista?

Ordenación

El funcionamiento como puntero de una lista tiene también sus repercusiones en el último aspecto que vamos a estudiar de esta tipología de elementos: la **ordenación**. Python ofrece **dos maneras de ordenar una lista**:



De manera **permanente**

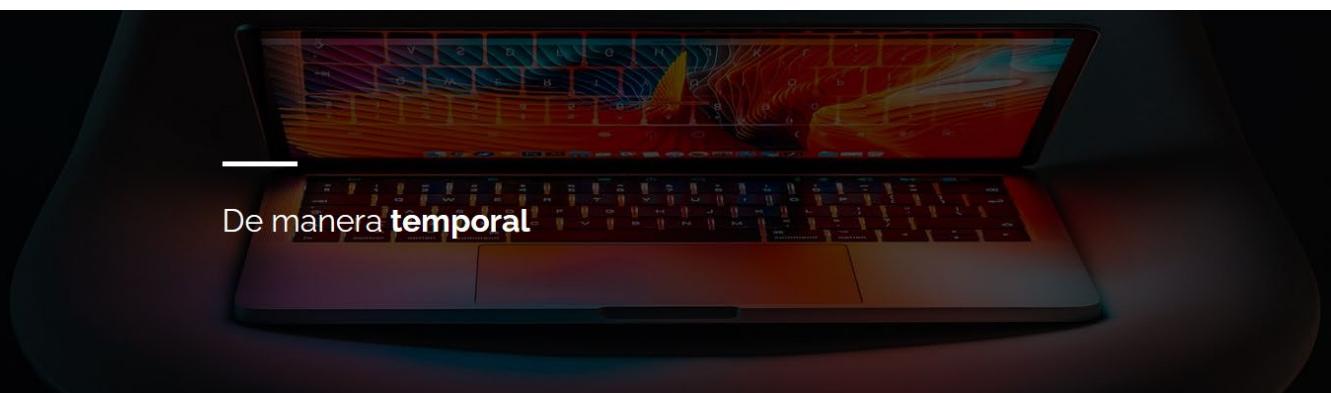
A través del método `sort()`. Si lo aplicas sobre una lista esta se ordenará alfabéticamente o de menor a mayor, pero este cambio resultará definitivo (no se podrá volver al orden original). Presta atención a la imagen:

```
1 #Este es el listado inicial de empresas que han solicitado la subvención
2 empresas=["El Corte Inglés", "ACCIONA", "Endesa","Fagor", "HP"]
3 #Las ordenamos de manera permanente
4 empresas.sort()
5 print("Listado inicial",empresas)
```

La salida de este código es el que muestra la imagen:



Por tanto, hemos perdido el listado inicial, el que tenemos es el ordenado.



De manera temporal: mediante la función `sorted()`. De este modo, no se afecta a la posición en memoria de los datos (como sucedía anteriormente) y la lista original permanece invariable. Puedes comprobarlo en el siguiente código de la imagen:

```
1 #Este es el listado inicial de empresas que han solicitado la subvención
2 empresas=["El Corte Inglés", "ACCIONA", "Endesa","Fagor","HP"]
3 #Las ordenamos de manera temporal y mostramos la lista en pantalla
4 print("Listado ordenado",sorted(empresas))
5 #Imprimimos el listado inicial
6 print("Listado inicial",empresas)
```



A veces el orden que se obtiene con los métodos de ordenación de Python no se corresponde exactamente con el que cabía esperar. Esto sucede cuando se utilizan textos en mayúsculas y minúsculas, de ahí de que se recomiende en muchas ocasiones trabajar únicamente en minúsculas y emplear los recursos necesarios para mostrar determinados caracteres en mayúsculas.

Tuplas

Existen otros elementos muy parecidos a las listas que reciben el nombre de **tuplas**. De una manera muy sencilla, se trata de listas que son inmutables, es decir, que no se pueden modificar (recuerda que no existía una opción similar para las variables constantes). Una tupla se define mediante paréntesis y el mecanismo de acceso es similar al de las listas. Fíjate en la siguiente imagen:

```
1 #Un ejemplo de tupla
2 especificaciones=(300,150,"color blanco")
3 #El elemento en la posición 0 es 300; tenemos que seguir empleando corchetes
4 print(especificaciones[0])
5 #Si tratamos de hacer cambios el sistema devuelve un error
6 especificaciones[1]=200
```

Lo que sí resulta posible es la sobreescritura de la tupla original. Si cambias la línea 6 por `especificaciones=(150,200,"color negro")` Python no devolverá ningún error (compruébalo).

Conjunto

Python plantea otro mecanismo para la agrupación de objetos, denominado **conjunto**, con funcionalidad análoga a la de este elemento matemático. Como indican García et al. (2019), los conjuntos son «colecciones de objetos mutables, iterables y no ordenados». Se pueden definir de dos maneras:

Directamente

- **Directamente:** colocando los elementos entre llaves y separados por comas. En la siguiente imagen se recogen distintas definiciones de conjuntos con este enfoque. Fíjate en las salidas obtenidas:

The diagram shows a dark rectangular box containing Python code. To the right of the box, three light blue circles are connected by lines to the outputs of the code. The first output is labeled 'Conjunto 1', the second 'Conjunto 2', and the third 'Conjunto 3'.
The code is as follows:
1 conj_1={2,4,6,8} ◉ Conjunto 1
2 print(conj_1)
3 conj_2={2,4,4,6,8,8} ◉ Conjunto 2
4 print(conj_2)
5 conj_3={[2,4],[6,8]} ◉ Conjunto 3
6 print(conj_3)

En el primer conjunto (línea 1) no se aprecia ningún problema. Puedes comprobar que, en línea con lo indicado al describir sus características, cuando se muestra en pantalla los datos aparecen desordenados.

Los conjuntos no permiten elementos repetidos. Por ese motivo Python suprime los que son redundantes, por lo que conj_2 (definido en la línea 3) solo consta de 4 elementos.

Por último, los conjuntos no pueden contener objetos mutables como las listas o los diccionarios. Ese es el motivo de que aparezca un error cuando conj_3 (definido en la línea 5) se intenta mostrar en pantalla, ya que este conjunto está formado por dos listas. No se produciría ningún error si conj_3 quedase integrado por dos tuplas (compruébalo).

A través de la función set

Que transforma los elementos proporcionados en constituyentes de un conjunto. Empleando este mecanismo sí que podríamos obtener un conjunto a partir de un elemento mutable, tal y como se muestra en el siguiente código de la imagen:

```
1 conj_3=set([2,4,6,8])
2 print(conj_3)
```

La función `set()` no te permite incluir dos listas, como nos planteábamos en el ejemplo anterior. En realidad, lo que hace es extraer los números de la lista y los asigna a un conjunto, como si estuviéramos realizando la definición directa. Esta función resulta fundamental para poner inicializar un conjunto vacío (y no crear, de manera involuntaria, un diccionario que no tuviera elementos).

Antes hemos hablado de los diccionarios, ¿quieres profundizar en ellos?

Diccionario

¿Qué crees que es un diccionario en Python? Ahora lo descubrirás.



«Un **diccionario** en Python es una colección de pares clave-valor. Cada clave se conecta a un valor y podemos usar una clave para acceder al valor asociado a la misma. El valor de una clave puede ser un número, una cadena, una lista o incluso otro diccionario» (Matthes, 2020).

Los diccionarios constituyen uno de los **elementos más potentes** de este lenguaje en el contexto del tratamiento de los datos. Se trata de estructuras mutables, para las que no resulta posible la repetición de claves, ha de ser **única**, dado que es el componente que sirve para acceder al valor asociado.

Retomaremos una lista que definimos anteriormente, evaluacion, para ver que resulta mucho más interesante trabajar con sus datos a través de esta nueva estructura:

```
1  evaluacion={"matemáticas":8.5,"inglés":6.5,"ciencias":7.8,"economía":5.5}
2  print(evaluacion)
3  print(evaluacion["matemáticas"])
```

Fíjate en que los diccionarios se definen mediante **llaves** (como sucede con los conjuntos), aunque tienes que colocar dos puntos entre la clave y el valor (al principio resulta muy común confundirse y emplear el signo igual). La línea 2 muestra en pantalla el contenido del diccionario evaluacion y la 3 produce la salida de 8.5, que es el valor asociado a la clave "matemáticas".

La **inicialización a cero** de un diccionario se realiza con una llave vacía. A partir de aquí, la incorporación de nuevos pares resulta muy sencilla. En la siguiente imagen te planteamos la definición de evaluacion con este segundo enfoque, que permite modificar la estructura inicial a medida que se originan nuevos datos. En nuestro caso, por ejemplo, cuando conocemos las notas alcanzadas en cada asignatura:

```
1 evaluacion={}
2 evaluacion["matemáticas"]=8.5
3 evaluacion["inglés"]=6.5
4 evaluacion["ciencias"]=7.8
5 evaluacion["economía"]=5.5
6 print(evaluacion)
7 print(evaluacion["matemáticas"])
```

Las salidas de este código son similares a las del caso anterior, como cabía esperar.

```
{'matemáticas': 8.5, 'inglés': 6.5, 'ciencias': 7.8, 'economía': 5.5}
8.5
```

Si, por ejemplo, nos hemos equivocado al transcribir las notas y tenemos un 6.0 en “economía” bastará con añadir una nueva línea y modificar el valor indicado:

```
6 evaluacion["economía"]=6.0
7 print(evaluacion)
```

```
{'matemáticas': 8.5, 'inglés': 6.5, 'ciencias': 7.8, 'economía': 6.0}
```

Si deseamos eliminar un par clave-valor tendremos que emplear la sentencia `del`, indicando la clave a borrar. Supongamos, por ejemplo, que queremos prescindir del par “ciencias-7.8”. El código requerido es:

```
7  del evaluacion["ciencias"]
8  print(evaluacion)
```

La salida en pantalla nos permite comprobar que, en efecto, ha desaparecido la referencia a la clave "ciencias"; únicamente quedan las otras tres asignaturas:

```
{'matemáticas': 8.5, 'inglés': 6.5, 'economía': 6.0}
```

¿Quieres conocer otra forma de acceder a un valor a partir de su clave?

Método get()

Existe otra forma, muy interesante, de **acceder a un valor a partir de su clave**, que consiste en emplear el método `get()`. Al código ya existente añadimos una línea mediante la que cargamos el valor asociado a la clave "matemáticas" como muestra la siguiente imagen:

```
8  valor_mates=evaluacion.get("matemáticas")
9  print(valor_mates)
```

Efectivamente, la salida del código es 8.5. El interés de este método (muy empleado, en sus distintas variantes, en numerosos lenguajes de programación) es que nos puede servir para **tratar posibles errores**. Ten en cuenta que, en esta introducción a Python, hemos omitido las referencias al manejo de las excepciones, dado que la importancia de estos elementos se manifiesta, sobre todo, en contextos de aplicaciones ya en producción.

Seguramente el error más común en este contexto sea llamar a una **clave inexistente**. Si, por ejemplo, escribes `print(evaluación["informática"])` el sistema te devolverá un error, dado que no existe ninguna clave con ese nombre. La manera de prevenir este problema consiste en emplear un segundo argumento de `get()`, en el que indicaremos el mensaje a mostrar caso de que invoquemos una clave no disponible. Esto hará, además, que el programa no se detenga. Te planteamos que analices este código y lo pruebes. Comprobarás que, en

efecto, advierte del error en la llamada a la clave “informática”, pero continúa mostrando el siguiente valor solicitado como muestra la imagen:

```
1 evaluacion={}
2 evaluacion["matemáticas"]=8.5
3 evaluacion["inglés"]=6.5
4 evaluacion["economía"]=6.0
5 valor_informatica=evaluacion.get("informática","No existe nota de informática")
6 valor_ingles=evaluacion.get("inglés","No existe nota de inglés")
7 print("Informática:",valor_informatica)
8 print("Inglés:",valor_ingles)
```

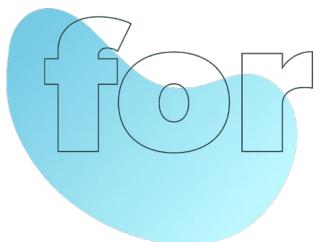
Control del flujo

¿Te han quedado claras las utilidades del diccionario en Python? Ahora conocerás otro de los aspectos fundamentales que tienes que identificar en cualquier lenguaje de programación. Se trata del correspondiente a las estructuras para el **control del flujo** de la aplicación. ¿Sabes cuál es su utilidad? Su aplicación permitirá la **ejecución de grupos de sentencias**, ya sea de manera iterativa o en base a unos condicionantes establecidos. Avanza en el contenido para conocer ambas opciones.

Control de flujo iterativo

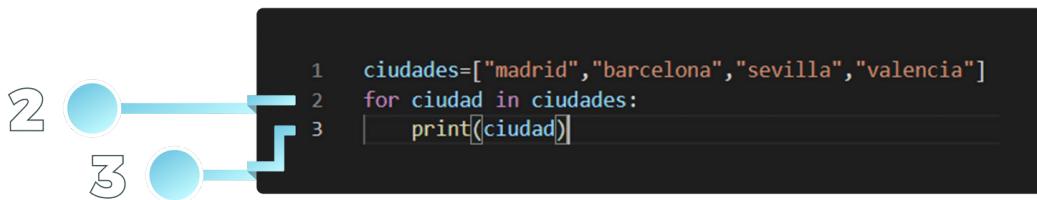
El **control de flujo iterativo** se lleva a cabo con los bucles `for` y `while`. Estos elementos se emplean para ejecutar una parte del código de manera repetitiva, aunque de dos maneras diferentes.

Ahora sabrás más sobre cada uno de los bucles.



El primero de los bucles indicados, `for`, se suele emplear para **recorrer los objetos iterables** que ya has estudiado, es decir, las cadenas, las listas, los conjuntos y los diccionarios. Recuerda que, en su momento, creamos la lista ciudades y que vimos que su salida en pantalla no era muy satisfactoria o que resultaba tedioso acceder a

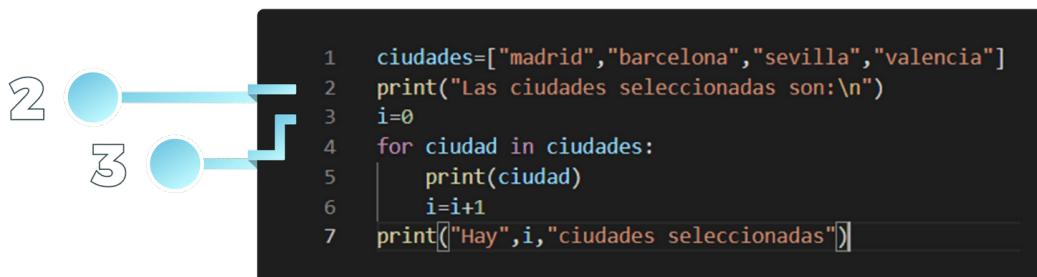
cada uno de los elementos que integraban dicha lista. Con `for` la impresión de dichos elementos es muy sencilla. Fíjate en la imagen y pon atención en dos hechos:



- **Línea 2:** en primer lugar, hemos utilizado una **variable auxiliar**, `ciudad`, para recorrer toda la lista. Cada vez que Python entra en `ciudades`, asocia el valor que corresponda a `ciudad`, y este es el que se muestra en pantalla.
- **Línea 3:** por otro lado, ten en cuenta que en la línea 3 hay un **sangrado**, que no se puede suprimir en ningún caso. Recuerda que Python emplea la indentación o el sangrado de líneas para diferenciar distintas partes de los programas (esto dotaba al lenguaje de una mayor legibilidad). Si empleas un editor de texto o un IDE no tendrás problemas en trabajar con los sangrados, ya que el sistema te irá recomendando el formato a seguir y, en su caso, te advertirá de que lo apliques.

En cualquier caso, recuerda colocar los **dos puntos al final de la sentencia** que comienza por `for`; si no lo haces, el sistema no te aplicará el sangrado y cuando lo ejecutes, además, te dará un error.

Vamos a modificar el código anterior para que puedas visualizar mejor esa estructura de la aplicación a la que acabamos de hacer referencia. Fíjate en la imagen y pon atención en dos de las líneas:



- **Línea 2:** antes de iniciar el bucle, imprimimos en pantalla un mensaje y hacemos un salto de línea a su conclusión empleando `\n`. De ese modo, separamos este texto del listado de ciudades en sí.
- **Línea 3:** ponemos a cero un contador, `i`, a fin de que compruebes que, en efecto, el bucle se ejecuta 4 veces (al final de la aplicación el valor de `i` es 4). En este caso sencillo te bastaría con emplear `len(ciudades)` para determinar el número de ciudades que hay en la lista.



La línea 6 se puede sustituir por `i+=1`. Haz la prueba y verás que funciona correctamente. Esta abreviatura simplifica la escritura de códigos en los que se implementan numerosos contadores.

while

El bucle `while()` también permite ejecutar una parte del código, pero solo mientras se **cumpla** alguna condición. Un ejemplo clásico es un simple programa para contar de 1 hasta 10. Fíjate en la imagen:

```
1  contador=1
2  while contador<=10:
3      print(contador)
4      contador+=1
```

Al comienzo del programa el sistema comprueba si la variable auxiliar, contador, es menor o igual que 10. Es el caso (se inicializa en 1), así que entra en el bucle, imprime el valor de la variable y, seguidamente, la incrementa en 1. Vuelve a la línea 2, realiza la comprobación y continúa hasta que contador vale 11. En ese momento no se cumple la condición establecida y el programa concluye.

Cuando estudiaste el funcionamiento de las listas viste que la aplicación de `remove()` a una lista con elementos repetidos no resulta válida para eliminar todos los redundantes. La forma correcta de llevarlo a cabo es a través de un bucle `while()`, como puedes ver a través del siguiente código que tienes en la imagen:

```
1  amigos=["pedro","juan","antonio","juan","miguel","fernando","juan"]
2  #Queremos eliminar todos los "juan" de la lista. Aplicamos remove y comprobamos el resultado
3  amigos.remove("juan")
4  print("Primer borrado:",amigos)
5  #Solo se borra el primer "juan". Volvemos a incorporar a "juan" al listado
6  amigos.insert(1,"juan")
7  print("Lista inicial:",amigos)
8  #Aplicamos el bucle de borrado
9  while "juan" in amigos:
10     amigos.remove("juan")
11  print("Borrado completo:",amigos)
```

La condición asociada a `while()` puede ser la que más nos interese: verás códigos en los que se hace referencia a que una parte del programa se ejecute mientras no se cumpla una condición.

Control de flujo condicional

El **control de flujo condicional** se realiza mediante la sentencia `if`, que permite realizar un análisis del estado de un programa y, en base a ello, lanzar la actuación correspondiente.

A través del siguiente código le pedimos a una persona usuaria que introduzca dos números, para lo que se requiere la función `input()`. El sistema analiza las dos variables y, de momento, comprueba si el primer número introducido es mayor que el segundo. De ser así, lanzará un mensaje confirmatorio.

```
1  mensaje1="Introduzca la primera variable:"  
2  mensaje2="Introduzca la segunda variable:"  
3  variable1=input(mensaje1)  
4  variable2=input(mensaje2)  
5  if(variable1>variable2):  
6      print("La primera variable,",variable1,", es la mayor")
```

Haz la prueba introduciendo un primer número que sea mayor que el segundo. Verás que la salida es correcta. Si proporcionas un primer número menor que el segundo, el sistema no hará nada. Para que siga comprobando las dos opciones restantes necesitará `elif` (para el caso que no se cumpla la condición anterior y se desee evaluar otra) y `else` (para el caso que no se cumpla ninguna condición previa).

El programa completo quedaría tal y como muestra la imagen:

```
1  mensaje1="Introduzca la primera variable:"  
2  mensaje2="Introduzca la segunda variable:"  
3  variable1=input(mensaje1)  
4  variable2=input(mensaje2)  
5  if(variable1>variable2):  
6      print("La primera variable es la mayor:",variable1)  
7  elif(variable1==variable2):  
8      print("Las dos variables son iguales")  
9  else:  
10     print("La segunda variable es la mayor:",variable2)
```

Ten en cuenta que la condición lógica de igualdad, para comprobar si las dos variables son iguales, requiere del empleo de dos signos iguales: `==`. Si utilizas solo uno obtendrás un error.

Funciones propias

Concluiremos este breve repaso de la programación en Python tratando la elaboración de **funciones** propias. A lo largo del contenido anterior has conocido funciones nativas, es decir, implementadas en el lenguaje, como `print()`, `sort()`, etc. Aunque existen numerosos desarrollos proporcionados por Python, resultará habitual que necesites **escribir tu propio código** para realizar tareas más específicas, sobre todo cuando son muy repetitivas.

Para **definir una función** tendrás que emplear la palabra clave `def`, que advierte a Python de lo que estás generando. El siguiente código permite elaborar una función muy simple, que devuelve el saludo de bienvenida cuando la llamas. Fíjate en la imagen:

```
1  def hola():  
2      print("¡Hola mundo!")
```

Si en la siguiente línea escribes `hola()` y ejecutas el código se mostrará el mensaje de bienvenida.

```
1 def hola():
2     print("¡Hola mundo!")
3 hola()
```

Un aspecto muy interesante de las funciones es que les puedas pasar **argumentos**, trabajes con ellos y luego te devuelvan los resultados. Por ejemplo, imagina que en un determinado programa requieres calcular en numerosas ocasiones el cociente de la suma de dos números y la resta de ambas. Lo más razonable consistirá en **definir una función** que realice esa operativa y recoger la llamada en el código cada vez que haga falta:

```
1 #Llamaremos a nuestra función como func_esp. Requerirá como argumentos los dos números
2 def func_esp(a,b):
3     resultado=(a+b)/(a-b)
4     #A la conclusión del cálculo devolvemos el resultado obtenido
5     return resultado
```

Para comprobar que, en efecto, funciona, escribiremos un código en la parte inferior, desde el que llamaremos a `func_esp` para que realice el cálculo:

```
1 #Llamaremos a nuestra función como func_esp. Requerirá como argumentos los dos números
2 def func_esp(a,b):
3     resultado=(a+b)/(a-b)
4     #A la conclusión del cálculo devolvemos el resultado obtenido
5     return resultado
6 var1=54
7 var2=34
8 res=func_esp(var1,var2)
9 print(res)
```

La salida es 4.4, que corresponde a la operación $(54+34)/(54-34)$.

$(54+34) \ (54-34)$

4.4

Es importante que tengas en cuenta que el orden en el que pasamos los argumentos es fundamental. La primera variable que ponemos en la llamada es la que `func_esp` interpretará como `a` y la segunda será `b`. Haz la prueba de invertir el orden (primero `var2` y luego `var1`) y comprobarás que el resultado cambia.

Lo normal y recomendable es que guardes la función en un archivo independiente y que la parte ejecutable la consignes en otro archivo, desde el que llamarás a la función. Para ello, requerirás de una sentencia que conocerás en el siguiente epígrafe, cuando estudies la importación de las librerías.

Recapitulación parcial

Has definido en qué consiste un lenguaje de programación y has identificado varios de sus tipos. Una de sus características principales es el paradigma en el que se basan, a partir del que has hecho una clasificación de algunos lenguajes de programación.

Python es uno de los más populares en la actualidad, tiene un gran potencial para el tratamiento integral de los datos y es uno de los más sencillos de aprender.

Has enumerado una serie de buenas prácticas y de recomendaciones en programación que te ayudarán a la hora de trabajar con Python.

También has descubierto los pasos para su instalación en diversos sistemas operativos y cómo empezar a trabajar con él, para lo que has conocido sus principales variables y algunas de sus funciones propias. Además, has valorado la importancia de trabajar con un editor de texto y las utilidades de un diccionario en Python, que es uno de los elementos más potentes para el tratamiento de los datos.

2.

Librerías de Python para Machine Learning



¿De qué te servirán todos los conocimientos que acabas de adquirir sobre Python? ¡Efectivamente! Para aplicarlos al Machine Learning. ¿Cómo? Avanza en el contenido.

Imagina que dispones de un **conjunto de datos**, obtenidos tras la medición de ciertos acontecimientos o relacionados con una situación concreta, por ejemplo, los salarios profesionales medios clasificados en función del puesto desempeñado. ¿Qué **procedimientos de Python**, de los descritos en el epígrafe anterior, crees que resultarían más interesantes para extraer información de dichos datos? ¿Qué **herramientas** echas en falta, en base a tu experiencia personal o profesional, para empezar a plantear un análisis de estos datos? Seguramente, en lo primero que pensarías es en disponer de instrumentos que posibiliten la **visualización de las mediciones realizadas** para poder detectar, rápidamente, algún patrón de comportamiento. También es muy probable que consideres adecuado plantear determinadas **herramientas estadísticas**.



¿Qué pensarías si te dijéramos que muchos de los procedimientos de análisis se encuentran **ya desarrollados** por personas expertas y están a disposición de las personas usuarias de Python? Estas operativas de análisis se encuentran implementadas en repositorios que reciben el nombre de **librerías**. A lo largo de este epígrafe, conocerás cómo se emplean y estudiarás las más utilizadas en el ámbito de Machine Learning. ¡Áñimate a descubrir el potencial de todos estos paquetes informáticos!

Antes de profundizar en el estudio de las librerías de Python, resulta idóneo que sepas diferenciar entre **tres conceptos** que emplearemos de manera frecuente, tanto de manera explícita como implícita. En contextos informales se emplean como sinónimos, pero su alcance es muy diferente:

- **Dato:** en primer lugar, debes considerar el concepto de **dato**, que hace referencia a un hecho, registrado de algún modo, que no tiene significado en sí mismo.



Una serie de valores numéricos consignados en una lista de Python, sin mayor referencia, constituyen un conjunto de datos.



- **Información:** el segundo de los conceptos es el de **información**, que se identifica como un dato o conjunto de datos que sí posee significado para, al menos, una persona.



Siguiendo con el mismo ejemplo, si se indica que los valores registrados en la hoja de cálculo son los ingresos diarios en un determinado establecimiento, se pasa a disponer de una información interesante sobre la evolución de ese negocio.



- **Conocimiento:** finalmente, el tercer grado de significación se corresponde con el del **conocimiento**, que es la constatación de que, en principio, una persona puede emplear la información a su disposición para conseguir un determinado objetivo, en base a la experiencia y formación acumulada.



En la misma línea del ejemplo anterior, el conocimiento de una persona especialista en el sector le permite emitir un juicio sobre la previsible rentabilidad del negocio considerando la información sobre los ingresos obtenidos. Esto puede servir a posibles personas inversoras para tomar una decisión respecto a invertir o no en dicho negocio.

Resulta evidente la importancia que tendrá para las organizaciones la generación y, sobre todo, el mantenimiento del conocimiento sobre el sector en el que operan y sobre los procesos productivos que lleva a cabo. Esto implica un paso previo, que es el de la aplicación de técnicas para obtener información a partir de los datos, de ahí la necesidad de implementar herramientas encuadradas en el contexto de la **Data Science**.



La cuestión que, en el fondo, se plantea en torno al Machine Learning es si ese conocimiento, tal y como se acaba de describir, resulta característico únicamente de los seres humanos o si podemos pensar en la factibilidad de un **conocimiento adquirido por las máquinas** y dispositivos electrónicos. Retomaremos esta cuestión en el siguiente epígrafe.

Librerías

Como ya sabes, el empleo de **librerías de software** facilita, de manera muy relevante, el proceso de transformación de datos en información. Para comprender el verdadero alcance de estas herramientas partiremos de la siguiente definición.



Una **librería** «es un archivo o conjunto de archivos que se utilizan para facilitar la programación. Las librerías, también llamadas "frameworks", consisten en archivos de código a los que llamamos al principio de la página» (Aprende Web, s.f.).

En realidad, este término corresponde con una traducción errónea del original en inglés, **library**, es decir, biblioteca. De hecho, resulta común encontrar referencias a la denominada «Biblioteca estándar de Python», sobre la que hablaremos un poco más adelante.

Al igual que una biblioteca física, una **librería informática** consiste en una ubicación, en este caso virtual, en la que se disponen determinadas implementaciones codificadas en el lenguaje de programación que corresponda. Ten en cuenta que una librería tiene un enfoque muy específico porque se orienta a la resolución de un problema acotado, por tanto, no posee una concepción generalista.

El empleo de librerías o bibliotecas se enmarca en uno de los principios básicos de la informática, el de **reutilización** (*reusability*) de código, que plantea que una parte o un programa completo ya existente, que cumpla de manera satisfactoria con las especificaciones a alcanzar, se debe emplear en el desarrollo de una nueva aplicación. Recuerda que dentro del paradigma de programación orientada a objetos se concedía una significativa importancia a la **abstracción**, que implicaba la supresión de la redundancia de ese código que se repetía en diferentes partes de un programa. Las bibliotecas de software facilitan el cumplimiento de este planteamiento, al permitir agrupar de manera coherente las distintas operativas desarrolladas.

Resulta evidente que el empleo de una librería posee numerosas **ventajas**, entre las que destacan las siguientes:



- **Facilidad:** incorporan métodos y funciones que, por lo general, resultan muy sencillos de utilizar. El propio IDE te irá indicando los atributos disponibles y, además, existe información en las páginas web de los desarrolladores para aclarar las dudas. Aparte,

estas funciones son complejas de programar, lo que evita un trabajo de cierto nivel a las personas que están desarrollando el modelo en cuestión.

- **Flexibilidad:** los códigos se encuentran disponibles en abierto, lo que proporciona a personas expertas en la materia la libertad necesaria para adaptarlos si fuera necesario (no es algo habitual).
- **Credibilidad:** los códigos disponibles en las librerías han sido muy testados por las personas desarrolladoras y por aquellas que han trabajado con ellos y que habrán advertido, en su momento, de posibles errores.
- **Respaldo:** existen numerosas comunidades de personas que desarrollan aplicaciones que suelen crear e intervenir en foros para alertar de posibles problemáticas e incorporar soluciones.
- **Globalidad:** existen librerías en todos los ámbitos de trabajo del Machine Learning.

Las librerías están formadas por un conjunto de **módulos** (trozos de programas centrados en una única finalidad) que, como acabas de comprobar, se agrupan para dar respuestas específicas. El número y la agrupación de dichos módulos puede ser muy variada.

Por lo general, se requiere un proceso previo de **importación** para que Python puede utilizar los módulos que forman parte de una librería. La función genérica que debes emplear para ello es `import`, que permite cargar en la carpeta correspondiente el módulo o el paquete (que son carpetas que contienen en su interior otros módulos y paquetes) que se trate.

¿Quieres saber cómo hacerlo? ¡A continuación, encontrarás un vídeo que te lo explicará!

Funcionamiento de una librería Python



¿Quieres conocer el funcionamiento de una librería Python? En el siguiente vídeo, Miguel Ángel Pino, profesor en el Departamento de Economía Financiera y Dirección de Operaciones de la Universidad de Sevilla y autor de este contenido, expone un breve ejemplo para que sepas cómo funciona y las posibles actuaciones que se pueden llevar a cabo con `import()`.

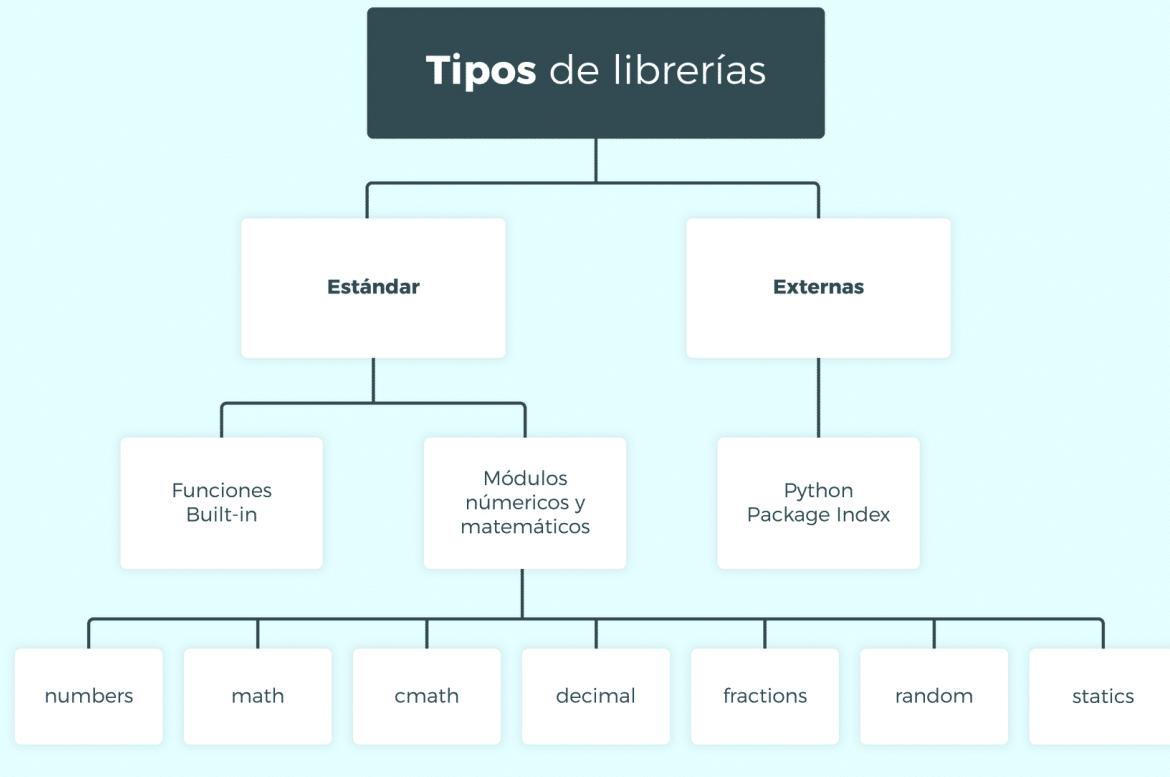
Cuando le indicamos a Python que importe un determinado elemento, el sistema lo buscará, en primer lugar, en el directorio en el que estés trabajando. Si no lo encuentra, se dirigirá a dos carpetas propias de la instalación del lenguaje, que son Lib (inicial de librería) y Lib/site-package, en las que ya hay, por defecto, un destacado número de módulos y paquetes, como tienes en la siguiente imagen del directorio Lib:

| | | | | | | | |
|-----------------|------------------|--------------|-------------|--------------|---------------|-------------|-----------|
| __pycache__ | wsgiref | bdb | filecmp | mimetypes | quopri | sunau | zipfile |
| asyncio | xml | binhex | fileinput | modulefinder | random | symbol | zipimport |
| collections | xmlrpc | bisect | fnmatch | netrc | re | syntable | |
| concurrent | zoneinfo | bz2 | formatter | nntplib | reprlib | sysconfig | |
| ctypes | _future_ | calendar | fractions | ntpath | rlcompleter | tabnanny | |
| curses | _phello_foo | cgi | ftplib | nturl2path | runpy | tarfile | |
| dbm | _aix_support | cgitb | functools | numbers | sched | telnetlib | |
| distutils | _bootlocale | chunk | genericpath | opcode | secrets | tempfile | |
| email | _bootsubprocess | cmd | getopt | operator | selectors | textwrap | |
| encodings | _collections_abc | code | getpass | optparse | shelve | this | |
| ensurepip | _compat_pickle | codecs | gettext | os | shlex | threading | |
| html | _compression | codeop | glob | pathlib | shutil | timeit | |
| http | _markupbase | colorsys | graphlib | pdb | signal | token | |
| idlelib | _osx_support | compileall | gzip | pickle | site | tokenize | |
| importlib | _py_abc | configparser | hashlib | pickletools | smtplib | trace | |
| json | _pydecimal | contextlib | heapq | pipes | sndhdr | traceback | |
| lib2to3 | _pyio | contextvars | hmac | pkgutil | socket | tracemalloc | |
| logging | _sitebuiltins | copy | imaplib | platform | socketserver | tty | |
| msilib | _strptime | copyreg | imghdr | plistlib | socketserver | turtle | |
| multiprocessing | _threading_local | cProfile | imp | poplib | sre_compile | types | |
| pydoc_data | _weakrefset | crypt | inspect | posixpath | sre_constants | typing | |
| site-packages | abc | csv | io | pprint | sre_parse | uu | |
| sqlite3 | aifc | dataclasses | ipaddress | profile | ssl | uuid | |
| test | antigravity | datetime | keyword | pstats | stat | warnings | |
| tkinter | argparse | decimal | linecache | pty | statistics | wave | |
| turtledemo | ast | difflib | locale | py_compile | string | weakref | |
| unittest | asynchat | dis | Izma | pyclbr | stringprep | webbrowser | |
| urllib | asyncore | doctest | mailbox | pydoc | struct | xdrlib | |
| venv | base64 | enum | mailcap | queue | subprocess | zipapp | |

Tipologías de librerías

Ahora que ya sabes en qué consisten las librerías, qué ventajas supone trabajar con ellas y cómo se importan, es el momento de que sepas qué tipos existen.

En Python se distinguen **dos tipologías de librerías**. Dentro de cada una de ellas se encuentran diversos **componentes**. Presta atención al siguiente esquema para que puedas enumerarlos rápidamente:



Como acabas de comprobar, hay dos **tipologías**. ¿Quieres saber en qué se diferencian?

- **Librería estándar:** en el primer grupo, se encuentra la nativa o **estándar**, que comprende un conjunto de módulos y distintos paquetes (todos ellos escritos en C o en el propio lenguaje Python), que se encuentran **incorporados** y se instalan con la distribución oficial del lenguaje.
Esta librería o **biblioteca estándar**, como resulta más común identificarla, consta de un elevado conjunto de componentes.



Aunque todos los componentes tienen gran importancia para las personas programadoras en Python destacaremos, en breve, los más relevantes en el contexto de los desarrollos relativos a Machine Learning. Para que te vayas adentrando en el tema, merece la pena que los revises haciendo clic en el siguiente enlace de Python [«La Biblioteca Estándar de Python»](#).

- **Librerías externas:** Python contempla, por otra parte, el uso de librerías externas, **desarrolladas por terceros.**

Seguro que quieras conocer cada uno de los elementos de los dos tipos de librerías que has conocido en el esquema. A continuación, podrás ir profundizando en ellos.

Funciones Built-in

Ya sabes que todos los **elementos** son relevantes para las personas programadoras en Python. Empezaremos por conocer, dentro de la biblioteca estándar, en qué consisten las funciones **Built-in**, es decir, incorporadas.

Las funciones **Built-in** son aquellas que corresponden a funciones que utiliza directamente el intérprete, sin que se requiera una llamada a través de `import()`.



¿Quieres conocer todas las funciones **Built-in**? Haz clic en el siguiente enlace y accederás a la sección Documentación de la página de Python donde recoge todas las **funciones Built-in**.

En la siguiente captura se muestra cómo, por ejemplo, la función `abs()`, que devuelve el valor absoluto de un número (el mismo número si es positivo o el opuesto si es negativo), se puede llamar directamente, sin requerir ninguna importación previa (el mismo caso que `print()`, como has estudiado anteriormente):

```
1 num=-5.6
2 num_abs=abs(num)
3 print(num_abs)
```

La mayoría de estas funciones se emplea de manera muy frecuente en el **tratamiento de datos**.

Módulos numéricos y matemáticos

Los módulos de la **biblioteca estándar** más empleados en Machine Learning son, sin duda, los **numéricos y matemáticos**. Bajo esta agrupación se encuentran los que descubrirás a continuación:

Módulo numbers

- **Módulo numbers:** se emplea fundamentalmente para trabajar con **clases numéricas abstractas**.

En la siguiente captura puedes comprobar cómo se requiere, previamente, la importación del módulo para poder aplicar sus métodos. La función `isinstance(argumento, Number)` se encuentra dentro del módulo numbers y devuelve un valor “True” si el argumento es un número, como es el caso de la imagen, (prueba a emplear un texto como argumento para comprobar que la respuesta es “False”).

```
1 import numbers  
2 num=7.8  
3 print(isinstance(num,numbers.Number))
```

Módulo math

- **Módulo math:** permite implementar **funciones matemáticas** de diversa índole, como combinatorias, logarítmicas, exponenciales, trigonométricas, hiperbólicas, etc. y trabajar con constantes especiales. Es importante que tengas en cuenta que este módulo no permite operativas con números complejos (esta es la tipología, por ejemplo, de la raíz cuadrada de un número negativo).

A modo de ejemplo, a continuación, tienes algunas de las **funciones** implementadas en math más conocidas y empleadas en nuestro ámbito de estudio:

- `math.ceil(x)` determina el primer número entero mayor o igual que x (por eso se dice que esta es la función «techo»).
- `math.floor(x)` devuelve el primer número entero menor o igual que x (se trataría, por tanto, de la función «suelo»).
- `math.sqrt(x)` proporciona la raíz cuadrada de x (este número ha de ser forzosamente positivo, como acabas de ver).
- `math.exp(x)` calcula el valor de e elevado a x (el valor e corresponde a la constante de Euler, 2,7182...).
- `math.cos(x)` devuelve el coseno del ángulo x, expresando en radianes.
- `math.pi` proporciona el valor de π hasta la precisión fijada.

Actividad práctica: función implementada en math

Inicializa en Python una variable con el valor 6.7 y determina, mostrándolo en pantalla, el «suelo» de este número. ¿Crees que funcionará el código que has escrito si inicializas la variable con -6.7? Piensa cuál es la respuesta y luego compruébalo.

Solución

El código para realizar la operación planteada es:

```
1 import math
2 num=6.7
3 print(math.floor(num))
```

La salida del script es, evidentemente, 6. Si cambias la segunda línea por num=-6.7 el código sigue funcionando, dado que este es un número negativo, no es complejo. En este caso la salida es -7.

Módulo cmath

- **Módulo cmath:** posibilita trabajar con **funciones matemáticas para números complejos**, es decir, los argumentos de dichas funciones pueden ser enteros, reales o complejos. Aparte de incorporar métodos especiales para realizar conversiones desde coordenadas polares, tiene implementadas funciones con la misma denominación que en el módulo math, pero que este contexto sí que se pueden aplicar a números complejos.

Actividad práctica: empleando el módulo apropiado

Genera un código en Python para comprobar que el cálculo de la raíz cuadrada de -25 produce un error cuando se trabaja con el módulo math. Modifica el código y calcula seguidamente dicha raíz cuadrada, mostrándola en pantalla. Haz que también se muestre en pantalla qué tipo de variable es la obtenida.

Solución

El código inicial podría ser similar al que te mostramos en la siguiente imagen:

```
1 import math  
2 num=-25  
3 print(math.sqrt(num))
```

La salida de este código da error: math domain error. Para calcular correctamente esta raíz cuadrada tienes que emplear el módulo cmath:

x x x x
x x x x
x x x
x x

```
1 import cmath
2 num=-25
3 sol=cmath.sqrt(num)
4 print(sol,type(sol))
```

En este caso la salida es $-5j$ (un número complejo sin parte real) y el sistema indica que, en efecto, se trata de una variable 'complex'.

Módulo decimal

- **Módulo decimal:** permite una mayor precisión en el **tratamiento de números reales**, proporcionando técnicas de redondeo más precisas. Se trata de un módulo muy específico, poco empleado por lo general en nuestro contexto de trabajo.

Módulo fractions

- **Módulo fractions:** posee interesantes funcionalidades para trabajar con **números racionales** (aquellos que pueden expresarse como el cociente de dos números enteros). La instancia `Fraction` es, sin duda, la más utilizada en este conjunto de desarrollos porque proporciona resultados muy útiles en función de los argumentos empleados. Como ejemplo te indicamos algunas de estas **operativas**:

- **Permite obtener la fracción irreducible de una dada:** en este caso, el primer argumento de `Fraction` ha de ser el numerador y el segundo el denominador, tal y como puedes comprobar en la imagen inferior. El resultado es `7/6` (la fracción irreducible de `35/30`). En este caso, para simplificar hemos optado por importar únicamente la instancia: la primera línea se podría haber sustituido, como anteriormente, por `import fractions` y en la segunda tendrías que indicar `fractions.Fraction(35, 30)`.

```
1 from fractions import Fraction
2 print(Fraction(35,30))
```

- **Convierte en número racional una cadena con formato de fracción:** todas las funcionalidades que facilitan la transformación de cadenas en números resultan de gran interés en el campo de la Data Science, como verás en su momento. En este caso la salida es `4/5` y el tipo de variable `fractions.Fraction`.

```
1 from fractions import Fraction
2 num=Fraction('4/5')
3 print(num,type(num))
```

- **Da dos enteros cuyo cociente es el número real dado como argumento:** proporciona dos enteros cuyo cociente coincide con el número real proporcionado como argumento. Esta operativa es similar a la de `as_integer_ratio()`, que ya estudiaste. En el ejemplo aportamos como argumento el número real 0.75. Al invocar `Fraction` el sistema devuelve un cociente entre enteros cuyo valor es 0.75, de ahí que la salida sea $\frac{3}{4}$:

```
1 from fractions import Fraction
2 num=Fraction(0.75)
3 print(num)
```

- **Módulo random:** sirve para generar **números pseudoaleatorios**, que son aquellos que se obtienen a partir de un algoritmo prefijado, de modo que los resultados obtenidos cambian en función del valor de partida o semilla (**seed**, en inglés). Como indica su nombre no se trata de números completamente aleatorios (han de ser impredecibles por definición), pero que sí que pueden constituir una interesante aproximación y ser de utilidad en numerosos contextos (entre los que se excluyen los relacionados con técnicas criptográficas).

Algunas de las **funciones** más empleadas de este módulo son las siguientes:

- `random.randint(x, y)` devuelve un número entero pseudoaleatorio entre x e y (ambos inclusive).
- `random.random()` genera un número real pseudoaleatorio entre 0 y 1.
- `random.uniform(x,y)` devuelve un número real pseudoaleatorio entre x e y (ambos inclusive).
- `random.choice(lista)` proporciona un elemento pseudoaleatorio de una lista definida con antelación (si la lista está vacía se genera un error).
- `random.sample(población, n, *, counts=None)` devuelve una lista con n elementos únicos (que no se repiten) de una determinada población (de ahí su utilidad para realizar muestreos aleatorios sin reemplazo).

Actividad práctica: combinación de números

Crea un código en Python para que te genere y muestre en pantalla una combinación (esperemos que ganadora) de BonoLoto. Recuerda que en este caso necesitas 6 números

entre 1 y 49, diferentes entre sí.

Solución

Existen numerosos enfoques para resolver el problema planteado. En la imagen tienes un código muy sencillo, que permite comprobar que ninguno de los números previos se encuentra ya en la lista de resultados antes de incorporar uno nuevo. En este caso hemos incorporado la función `sorted` para que los números se muestren ordenados (lo que te facilitará la tarea de llenar el boleto).

```
1 import random
2 lista=[random.randint(1,49)]
3 i=1
4 while i<6:
5     posible=random.randint(1,49)
6     if posible not in lista:
7         lista.append(posible)
8         i+=1
9 print("La futura combinación ganadora de la Bonoloto es:",sorted(lista))
```

Módulo statistics

- **Módulo statistics:** proporciona un amplio conjunto de funciones que permiten realizar **cálculos estadísticos**, de ahí su importancia tan significativa en los estudios de Machine Learning. En este campo se distinguen **dos grandes campos**:
 - **Promedios:** las funciones que permiten calcular **promedios** y medidas de tendencia central, como:
 - `statistics.mean()` que devuelve la media de un conjunto de datos.

- `statistics.geometric_mean()` para el cálculo de la media geométrica (la raíz n-ésima del producto de un conjunto con n datos).
 - `statistics.harmonia_mean()` que proporciona la media armónica (la inversa de la media aritmética de los recíprocos de los elementos del conjunto considerado).
 - `statistics.median()` para determinar la mediana de un conjunto.
 - `statistics.mode()` que calcula la moda o valor más frecuente de un conjunto integrado por valores discretos o nominales, es decir, que solo pueden tomar un número finito de valores.
- **Medidas de dispersión:** las funciones que permiten determinar **medidas de dispersión**, entre las que destacan:
- `statistics.pstdev()` que devuelve la desviación típica poblacional.
 - `statistics.stdev()` que calcula la desviación típica muestral.
 - `statistics.pvariance()` para determinar la varianza poblacional de un conjunto de datos.
 - `statistics.variance()` que calcula la varianza de una muestra de datos.



Es muy recomendable emplear la función `statistics.fmean()` en vez de `statistics.mean()` para el cálculo de la media de un conjunto de datos. Esta función transforma los números en reales y aplica el algoritmo, consiguiendo resultados de este modo de una manera más rápida. Comprueba, por ejemplo, que si calculas la media de un conjunto de números enteros con `mean` obtienes un valor entero mientras que si lo haces con `fmean` te devuelve una variable float.

Python Package Index

Python contempla el uso de **librerías externas** desarrolladas por terceras personas. Es más, el propio lenguaje potencia la existencia de estos módulos y paquetes, ya que se encuentra diseñado para garantizar la interoperabilidad.

El número de librerías externas existente es muy elevado, al ser tan amplio el campo de aplicaciones de Python. Las librerías más conocidas poseen su propia ubicación en Internet,

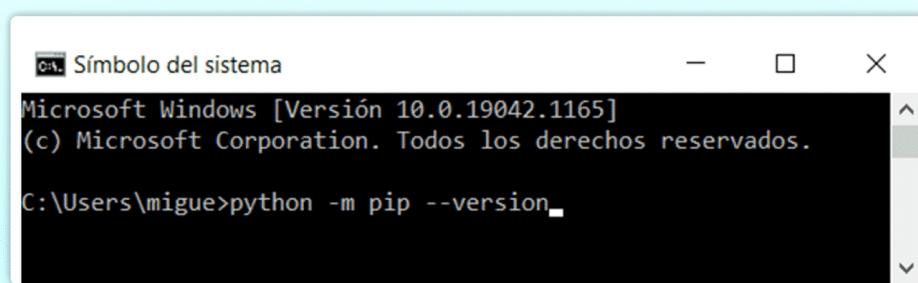
aunque, en cualquier caso, la gran mayoría se encuentra a disposición de las personas usuarias en el **Python Package Index** (normalmente conocido como **PyPI**).



Tienes que hacer clic en el siguiente enlace para acceder al Python Package Index. Una vez que estás ahí, si realizas una búsqueda para «machine learning» en el recuadro de búsqueda superior para localizar librerías especializadas en la materia obtendrás más de 10.000 resultados. Desde la propia pantalla de bienvenida de PyPI se muestra el número de proyectos alojados en la web (más de 325.000). Todo esto confirma la existencia de una elevada comunidad de personas que hacen desarrollos en Python, su destacado grado de compromiso y su actividad desbordante.

El motivo principal que justifica el **mantenimiento de distintas versiones de Python** en un equipo es el uso, por parte de una aplicación, de librerías externas que fueron diseñadas con versiones más antiguas y que no funcionan con los últimos desarrollos del lenguaje. Esta circunstancia resulta muy significativa, sobre todo, por la incompatibilidad entre las versiones 2. y 3.

La **instalación** de las librerías dispuestas en PyPI requiere de la herramienta `pip`. Si has instalado Python siguiendo alguno de los procedimientos descritos anteriormente, la tendrás habilitada (a partir de las versiones 2.7.9 y 3.4 viene por defecto con la instalación del lenguaje). Para comprobarlo, basta que abras en Windows el símbolo del sistema y ejecutes la siguiente instrucción que tienes en la imagen (en macOS deberás emplear `python3`):

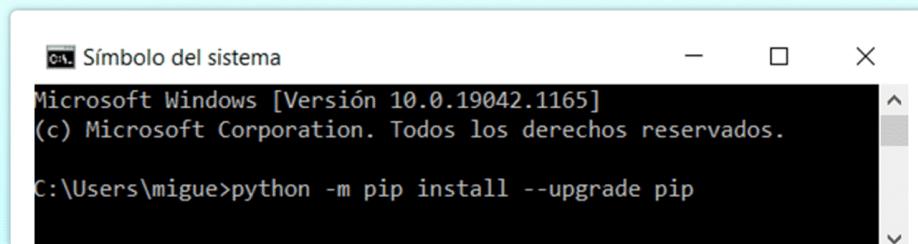


```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19042.1165]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\migue>python -m pip --version
```

La salida de esta instrucción confirmará que dispones de `pip` en tu sistema, ya que se indicará cuál es la versión instalada. Resulta muy recomendable que compruebes si se trata de la última

versión disponible tratando, sencillamente, de proceder a su actualización. Para ello debes escribir el texto que muestra la imagen:



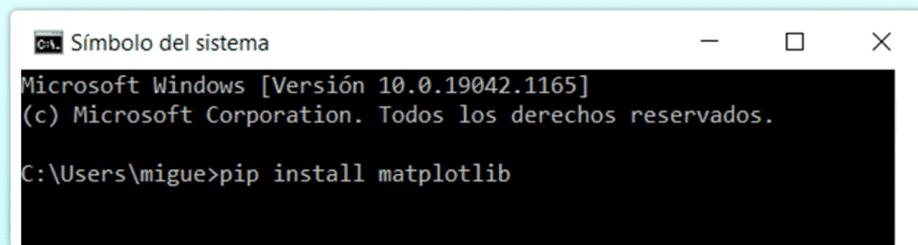
```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19042.1165]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\migue>python -m pip install --upgrade pip
```

La instalación de cualquier librería es muy sencilla con esta herramienta.

Ejemplo de instalación de librería Matplotlib

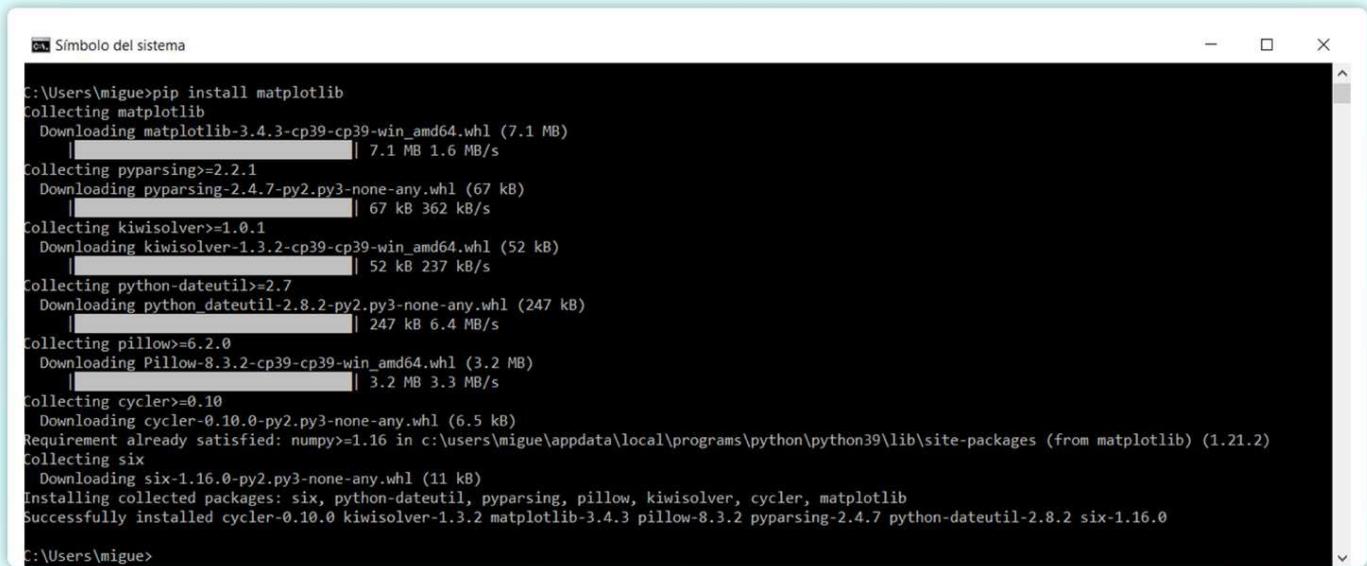
Imagina que deseas instalar **Matplotlib**, que es la biblioteca de referencia para la generación de gráficos en Python. Bastará con especificar el nombre de la librería en minúsculas en una instrucción como la siguiente:



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19042.1165]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\migue>pip install matplotlib
```

La salida informa de los elementos que se descargan (en este caso 6) y que la instalación se ha llevado de cabo de manera satisfactoria:



```
C:\Users\migue>pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.4.3-cp39-cp39-win_amd64.whl (7.1 MB)
    |████████| 7.1 MB 1.6 MB/s
Collecting pyparsing>=2.2.1
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
    |████████| 67 kB 362 kB/s
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.3.2-cp39-cp39-win_amd64.whl (52 kB)
    |████████| 52 kB 237 kB/s
Collecting python-dateutil>=2.7
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
    |████████| 247 kB 6.4 MB/s
Collecting pillow>=6.2.0
  Downloading Pillow-8.3.2-cp39-cp39-win_amd64.whl (3.2 MB)
    |████████| 3.2 MB 3.3 MB/s
Collecting cycler>=0.10
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Requirement already satisfied: numpy>=1.16 in c:\users\migue\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (1.21.2)
Collecting six
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, python-dateutil, pyparsing, pillow, kiwisolver, cycler, matplotlib
Successfully installed cycler-0.10.0 kiwisolver-1.3.2 matplotlib-3.4.3 pillow-8.3.2 pyparsing-2.4.7 python-dateutil-2.8.2 six-1.16.0
C:\Users\migue>
```

A partir de este momento podrás trabajar con esta librería de manera similar a las que integran la estándar de Python, realizando la importación de los módulos con la herramienta `import()`.

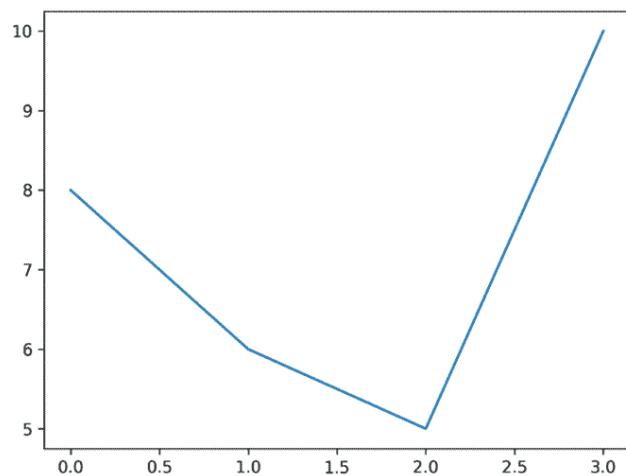
Ejemplo de importación de `pyplot` de Matplotlib

Si ingresas en **Visual Code Studio** puedes importar, por ejemplo, uno de los módulos más empleados de Matplotlib, **pyplot**, para así comprobar que ya dispones de la librería. En este caso vamos a realizar la importación de una manera alternativa a la que explicamos, pero también muy intuitiva.

En el código que tienes en la siguiente imagen hemos incluido, por ejemplo, las cuatro notas obtenidas por una persona en una serie de pruebas, a fin de representarlas en un gráfico. La última línea permite guardar la gráfica generada en un fichero .png en la misma carpeta de trabajo en las que estás guardando tus ficheros:

```
1 import matplotlib.pyplot as plt
2 notas=[8,6,5,10]
3 plt.plot(notas)
4 plt.savefig("notas_obtenidas.png",dpi=300)
```

El resultado de este script es:



+++
+++

+++
+++

Más adelante profundizarás en el manejo de esta librería y, en particular, de este módulo, mediante el que podrás obtener gráficas de nivel profesional. Lo que interesa de momento es que sepas que existen **otras opciones para instalar librerías** y que no necesitas abandonar Visual Studio Code para ello. Basta con que te desplaces a la terminal y en la parte inferior teclees la instrucción: `pip install nombre de la librería`:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscores6

PS C:\Users\migue\Downloads> & 'C:\Users\migue\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\migue\.vscode\extensions\ms-python.python\python-2021.9.1191016588\pythonFiles\lib\python\debugpy\launcher' '54372' '--' 'c:\Users\migue\Downloads\script2.py'
PS C:\Users\migue\Downloads> cd 'c:\Users\migue\Downloads' ; & 'C:\Users\migue\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\migue\.vscode\extensions\ms-python.python\python-2021.9.1191016588\pythonFiles\lib\python\debugpy\launcher' '63489' '--' 'c:\Users\migue\Downloads\script2.py'

PS C:\Users\migue\Downloads> pip install numpy
```

Ahora que sabes cómo instalar las librerías externas, te estarás preguntando cuáles son las más **recomendables** para los desarrollos en Machine Learning. No existe una respuesta cerrada al respecto debido el volumen tan elevado de alternativas existentes, aunque sí es posible identificar las más valoradas por las personas especialistas en este campo.

Librerías más valoradas

Para que puedas identificar las librerías más valoradas por las personas especialistas vas a hacer una revisión de las librerías considerando cuáles son las **tareas** de una persona que materializa proyectos en el contexto del Machine Learning y, a partir de ahí, concretar los módulos idóneos para satisfacer dichas tareas.

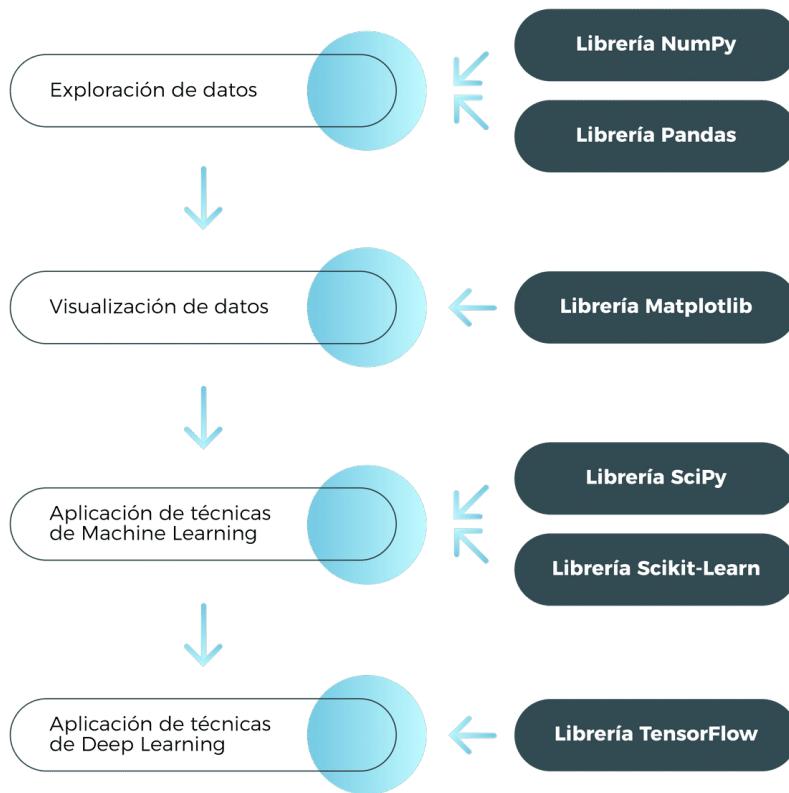
García et al. (2019) identifican **cuatro actuaciones fundamentales** en esta área y las relacionan con **cuatro librerías** muy populares. La siguiente figura, modificada respecto al planteamiento original, ilustra esta idea:



- **NumPy:** el término **NumPy** corresponde a las siglas de *Numerical Python*. Como señalan García et al. (2019), «es el principal paquete de Python destinado a la computación científica», es decir, al desarrollo de modelos matemáticos que tratan de simular comportamientos en las áreas de la ingeniería, ciencias sociales, investigación, etc. Su potencial queda probado por el hecho de que se trata de la **librería base** para el desarrollo de los restantes paquetes que tienes en la imagen.
- **Matplotlib:** **Matplotlib** es una librería focalizada en la obtención de gráficos, tanto bidimensionales como tridimensionales. Sus desarrollos originales trataban de emular el funcionamiento de MATLAB, que es un software privativo para cálculo matricial y análisis numérico, aunque se ha convertido en los últimos años en un paquete fundamental para la visualización de datos en Python.
- **SciPy:** **SciPy** proporciona rutinas para la realización de integraciones numéricas, interpolación, optimización, cálculos algebraicos y desarrollos estadísticos. Resulta muy habitual hablar, bajo un enfoque más amplio, del entorno o del **ecosistema SciPy**, que englobaría a los módulos de esta librería más a NumPy (dado que SciPy depende de esta librería para su ejecución) y a Matplotlib.
- **Pandas:** **Pandas** es otra librería muy popular en este contexto, ya que permite trabajar con series (que son una tipología de datos estructurados que pueden guardar conjuntos de datos de cualquier tipología) y con lo que se conoce como data frame (estructuras bidimensionales para guardar datos muy diversos, empleadas frecuentemente en el lenguaje R).

En la práctica estas cuatro librerías se suelen emplear de **manera complementaria**. De hecho, se encuentra en marcha una importante iniciativa encaminada a implementar mecanismos inclusivos en los ecosistemas científicos de Python, que se articula considerando las interconexiones entre dichos paquetes.

En la web de Solver (2021) se plantean **tres tareas genéricas y una específica** en el contexto del Machine Learning, con las correspondientes librerías asociadas, tal y como puedes comprobar en la siguiente imagen:



- **Exploración de datos:** persigue la mayor comprensión posible del problema al que nos enfrentamos, de ahí la idoneidad de contar con el apoyo de librerías como NumPy o Pandas.
- **Visualización de datos:** para la visualización, de nuevo se recurrirá al potencial de Matplotlib. Existen otras alternativas interesantes para dicha elaboración de gráficas, como pueden ser [Seaborn](#) (cuyo desarrollo se basa en Matplotlib) o [Bokeh](#) (orientada a la visualización interactiva de datos en páginas web).
- **Aplicación de técnicas específicas de Machine Learning:** en la web de Solver se identifican dos paquetes para esta tarea: SciPy y [Scikit-Learn](#), que contiene numerosos algoritmos para realizar tareas de clasificación, regresión, agrupación, etc. Esta librería contiene una funcionalidad muy relevante, que recibe el nombre de pipeline y que permite integrar diferentes secuencias encaminadas a transformar y reutilizar datos.
- **Aplicación de técnicas de Deep Learning:** dentro del amplio conjunto de técnicas de Machine Learning existen algunas que persiguen un modelado más cercano al funcionamiento del sistema nervioso humano y que se agrupan bajo la denominación genérica de Deep Learning. Estas técnicas se basan en el tratamiento de redes neuronales, siendo [TensorFlow](#), desarrollada por Google, la librería más empleada para su entrenamiento. El empleo de la librería se simplifica gracias a las API que se han elaborado para ello, entre las que destaca Keras. Otra alternativa es [PyTorch](#) que es otra librería muy empleada en el contexto del Deep Learning, desarrollada por Facebook.

Aunque no se trate de una librería propiamente dicha, merece la pena destacar en este apartado el interés de una distribución de Python como es **Anaconda**, muy empleada para cálculo numérico, análisis numérico y Machine Learning, como indica Martínez (2020). Una de sus ventajas es que incorpora, por defecto, la mayoría de las librerías descritas y que también facilita la implementación de las que resulten necesarias. En cualquier caso, lo más interesante de la distribución es que permite trabajar con diferentes entornos de trabajo, particularizados para los objetivos planteados, lo que resulta idóneo en el caso de abordar proyectos diferentes.

Recapitulación parcial

Has identificado la cantidad de librerías que existen en Python, funcionalidad que comparte con el resto de los lenguajes de programación, al ser trozos de programas desarrollados por terceras personas, suficientemente testados, que posibilitan una reutilización eficiente del código.

Python cuenta con una librería estándar. Tiene numerosos elementos, entre los que destacan las funciones Built-in o incorporadas y los módulos numéricos y matemáticos, como math, cmath o statistics.

Las librerías externas son aquellas desarrolladas por personas especializadas. Las más relevantes están en el Python Package Index. Los desarrollos en Machine Learning se facilitan con el empleo de las orientadas expresamente a la computación científica o a la visualización de datos. Las librerías más utilizadas en este contexto son NumPy, Matplotlib, SciPy o Pandas.

3.

Machine Learning. Introducción



¿Te has preguntado en alguna ocasión si los ordenadores o equipos similares podrán algún día pensar de manera semejante a como lo hacen los seres humanos? Al formular así esta pregunta quizás pienses que todavía no se comportan así (algo que es cierto), aunque ¿no es verdad que a veces no lo parece? Seguramente te hayas sorprendido con algún mensaje relacionado con algo que te interesaba que, de repente, aparece en tu equipo informático o móvil sin que tú lo hubieses solicitado. Todos estos mensajes aparecen como consecuencia de la aplicación de **algoritmos**, que es la forma en la que los equipos informáticos saben actuar. ¿Cómo funcionan estos algoritmos? A continuación, irás adentrándote en ellos.

Algunos **ejemplos** de los mensajes que pueden aparecer en tus dispositivos y que, probablemente, te hayas encontrado son:

- La aparición en la web de un comercio electrónico en el que entraste a comprar hace unos días de un **producto similar** al que adquiriste, a un precio muy atractivo.
- Una **noticia** que aparece en la pantalla de búsqueda de Google directamente relacionada con una materia que te apasiona.
- Una **notificación del móvil** informándote de retenciones en el camino habitual a tu lugar de trabajo justo cuando te dispones a montarte en el coche.



Estos ejemplos se encuentran muy lejos de lo que tenemos en mente cuando hablamos de una forma de pensar similar a la humana. Estos mecanismos de transmisión no tienen nada que ver con las dificultades existentes en ciertas películas de ciencia ficción para diferenciar entre individuos y esos replicantes o androides con forma y comportamiento enteramente humano que aparecían en *Blade Runner*.

Sin embargo, no cabe duda de que podemos identificar una **cierta tipología de inteligencia en las máquinas** que nos rodean y que esta inteligencia, por tanto, artificial, se está desarrollando a pasos agigantados en los últimos años.

¿Qué es la inteligencia artificial?



En el siguiente [vídeo](#) de *El Mundo* te ofrecen interesantes consideraciones históricas y reflexiones sobre la inteligencia artificial, que te servirán para centrarte en esta importante temática. A la conclusión de dicho vídeo deberás responder tres cuestiones sobre estos contenidos.

Conceptualización de inteligencia artificial

Antes de seguir profundizando en la inteligencia artificial, piensa en si podrías explicar de manera clara en qué consiste la inteligencia. Probablemente, te resulta difícil definir, con precisión, el concepto de inteligencia. Si preguntáramos a distintas personas al respecto, seguro que obtendríamos respuestas diferentes. Partiremos, por tanto, de una definición genérica y consensuada, que concretaremos con posterioridad para el ámbito de los dispositivos informáticos.



La **inteligencia** es la «capacidad de entender o comprender» (Real Academia Española, s.f., definición 1).

La RAE incorpora otros significados que permiten captar la **naturaleza poliédrica** de este término y que resultan de utilidad en nuestro campo de estudio: «capacidad de resolver problemas» (acepción segunda), «conocimiento, comprensión, acto de entender» (acepción tercera) o «habilidad, destreza y experiencia» (acepción quinta). Esta institución es sensible a los cambios que se producen en nuestra sociedad y en sus efectos en el léxico, por lo que ya incluye en su diccionario la siguiente definición:

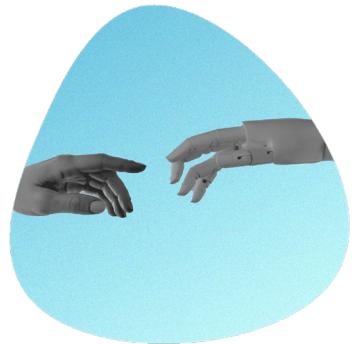


La **inteligencia artificial** es la «disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico».

La elaboración de estos programas requerirá que los dispositivos encargados de su ejecución tengan una cierta **capacidad de comprensión** y que hayan adquirido una cierta **destreza** en la materia de la que se trate (que es lo que intentaremos lograr a través de los entrenamientos).

Resulta posible precisar todavía más el concepto de inteligencia artificial a través de la definición que plantea Torres (2020) al respecto: «el esfuerzo para automatizar tareas intelectuales normalmente realizadas por humanos».

Desde mediados del siglo XX se ha trabajado en campos muy variados, aunque no de una manera tan continuada como pudiera parecer. Los avances en este ámbito no se han producido de una manera constante, sino a través de **fases** de expansión seguidas de otras de pérdida de interés, debido a que los resultados obtenidos no siempre han satisfecho a las personas usuarias. Ten presente que ese ideal de consecución de mecanismos de razonamiento totalmente similares a los seres humanos se encuentra muy lejano



En la actualidad nos encontramos en otro **período de desarrollo** en este contexto, como prueban los importantes avances en las tecnologías de reconocimiento automático de voz, detección de enfermedades, procesado de lenguaje natural y traducciones simultáneas, visión por computador o los desarrollos de nuevos materiales, entre otros.



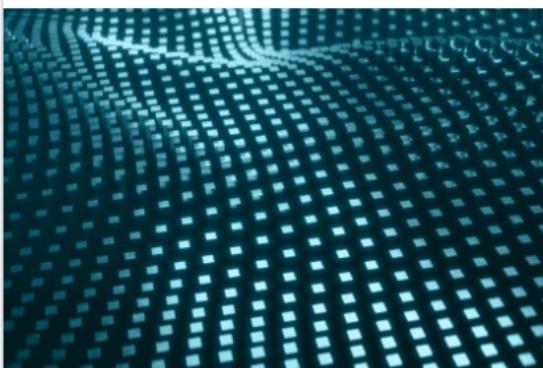
Las técnicas de inteligencia artificial posibilitan en la actualidad la generación de rostros falsos, prácticamente imposibles de diferenciar de otros verdaderos. Torres (2020) recomienda comprobarlo accediendo a la web ThisPersonDoesNotExist.com. Haz la prueba y te sorprenderás.

Probablemente la existencia de estos ciclos de «optimismo» y «pesimismo», como los denomina Torres (2020), respecto a los objetivos alcanzados por la inteligencia artificial se deban al hecho de que podemos distinguir **dos tipologías básicas** y que los principales avances solo se hayan dado en una de ellas:

- **Inteligencia artificial débil** (*artificial narrow intelligence*): es la que caracteriza a los sistemas informáticos que pueden realizar un conjunto muy limitado de tareas, probablemente incluso mejor que un ser humano, como el caso de mecanismos para el guiado milimétrico de maquinaria.
- **Inteligencia artificial fuerte** (*artificial general intelligence*): persigue el desarrollo de aplicaciones que puedan funcionar satisfactoriamente en diferentes ámbitos de trabajo. Esta capacidad de generalización, de poder realizar distintas cosas a la vez, en contextos muy cambiantes, es precisamente lo que caracteriza a los seres humanos.

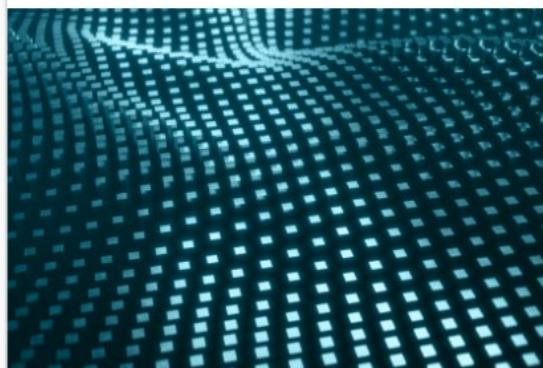
01

Inteligencia
artificial **débil**



02

Inteligencia
artificial **fuerte**



Los más relevantes y, prácticamente, únicos logros en inteligencia artificial se han conseguido en el primero de estos campos. Todavía no existen dispositivos a nuestro alrededor que puedan superar el test de Turing y no existen previsiones sobre cuándo se conseguirán los primeros resultados significativos en el segundo ámbito.



El test de Turing es una prueba planteada por su creador, el matemático Alan Turing, para tratar de analizar la inteligencia de las máquinas. La cuestión que trata de resolver este juego (dado que, en definitiva, se trata de eso) es si resulta posible que una máquina nos engañe haciéndose pasar por un ser humano cuando es sometida a una serie de preguntas.

Aunque las consideraciones sobre lo que se puede identificar como inteligencia artificial han cambiado con el tiempo y hoy en día las personas que investigan en este campo no están tan interesadas en ocultar la naturaleza artificial de sus modelos, resulta interesante que profundices en las características de dicha prueba. Si haces clic en el artículo [«¿Eres un robot? Aprueba el Test de Turing»](#) de Sapiens (Universidad Miguel Hernández de Elche), comprenderás cuáles son las principales limitaciones en esa «imitación del ser humano» en la que se trabaja desde la década de los 50 del siglo XX. Además, podrás comprobar si eres o no un robot (y no lo sabías).



Relación entre inteligencia artificial y Machine Learning

Ahora que eres capaz de definir qué es la inteligencia artificial y diferencias dos tipos básicos es el momento de que sepas cuál es su **relación con el Machine Learning**.

En la web de Aura Quantic (Fernández, s.f.) se recogen las conclusiones más significativas de un interesante seminario sobre inteligencia artificial y su relación con el Machine Learning. Para tratar esta relación resulta conveniente plantear otro mecanismo de agrupamiento de los diversos planteamientos presentes en el campo de la inteligencia artificial y así identificar distintas **tipologías** existentes:

- **Sistemas que piensan como seres humanos:** pueden automatizar la toma de decisiones, resolver problemas o aprender de manera autónoma. El ejemplo más relevante en esta categoría es el de las redes neuronales artificiales.
- **Sistemas que piensan de manera racional:** se centran en la reproducción de los razonamientos lógicos que realizan las personas. En este grupo se encuadran los denominados sistemas expertos.



Los **sistemas expertos** se comenzaron a desarrollar en la década de los 60 y, en cierto modo, se pueden considerar los primeros grandes avances en materia de inteligencia artificial. Es recomendable que profundices más en su conocimiento porque, en la actualidad, existen sistemas expertos con diseños excelentes que mejoran el funcionamiento humano en la toma de decisiones en contextos muy delimitados. Si haces clic en el siguiente enlace de Tecnologías Información, accederás a un artículo de su blog denominado [«Sistemas expertos: definición, aplicaciones y ejemplos»](#). Te ayudará a comprender en qué consisten y dónde se aplican.

- **Sistemas que actúan *como* humanos:** realizan tareas similares a las de cualquier individuo. En esta tipología se encuadran los robots.
- **Sistemas que actúan *de manera* racional:** tratan de imitar un comportamiento humano racional. Sería el caso de los agentes inteligentes, cuya concepción y visión ha ido cambiando en las últimas décadas.

Si te fijas en la clasificación expuesta los términos «como» y «de manera» están en cursiva. ¿Por qué? Para enfatizar que lo que se persigue es la **imitación** de los comportamientos de los seres humanos, de ahí la «apariencia» de inteligencia que muestra un móvil cuando te envía un mensaje que puede resultar de tu interés.

Bajo esta perspectiva, el Machine Learning (cuya traducción al castellano es aprendizaje automático) se centraría de manera específica en la reproducción de uno de esos comportamientos, en concreto el correspondiente a **cómo aprenden los seres humanos**. Se trata de la capacidad más significativa de los individuos. Esto implicará que el Machine Learning incorpore planteamientos tanto de la inteligencia artificial en sí como de las ciencias de la computación y de la neurociencia, que se focaliza en el estudio del sistema nervioso humano.



Ahora que tienes claro el concepto de inteligencia artificial y su relación con Machine Learning, a pesar de que a lo largo de toda la unidad didáctica ya has trabajado sobre este último, ya estás en disposición de definir este término de manera precisa.

Machine Learning

¿Qué es Machine Learning? Presta atención a la siguiente definición.



Machine Learning «es el subcampo de la inteligencia artificial que proporciona a los ordenadores la capacidad de aprender sin ser explícitamente programados, es decir, sin que estos necesiten que el programador indique las reglas que deben seguir para lograr su tarea, sino que la hacen automáticamente» (Torres, 2020).

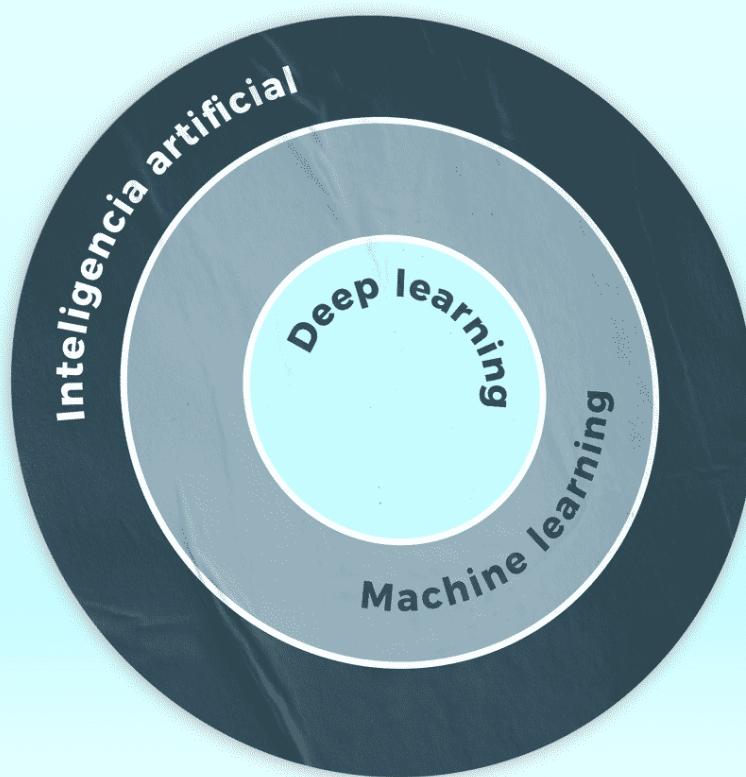
¿Crees que se trata de un concepto muy reciente? Lo cierto es que ya se empleaba a principios de la década de los 60, con un enfoque muy similar a la definición planteada. Fíjate en que estamos manejando un término muy conocido, que has empleado en multitud de ocasiones, como es aprender; pero también en este caso debemos precisar su significado.



Aprender consiste en «adquirir el conocimiento de algo por medio del estudio o de la experiencia» (Real Academia Española, s.f., definición 1).

En el contexto informático, el acto de aprendizaje se vincula a la generación y generalización de un conocimiento que se extrae a partir de un conjunto de experiencias.

Bajo estas consideraciones adquiere pleno sentido el esquema recogido en Torres (2020), y que puedes analizar a continuación en la siguiente imagen, en el que se plantea el Machine Learning como un **subconjunto** de la inteligencia artificial, aunque integrador de las distintas disciplinas que tratan de reproducir el comportamiento humano, por ejemplo, patrones de voz, visión computarizada, robótica, etc. Machine Learning está presente en cualquier aspecto de la inteligencia artificial, ya que todo sistema tiene que aprender, de una manera u otra, a realizar una actividad.



Dentro del Machine Learning se pueden distinguir, a su vez, distintas **técnicas** que ofrecen soluciones a distintos problemas de aprendizaje. Una de las que ha adquirido mayor relevancia

en las últimas décadas es el **Deep Learning** o aprendizaje profundo, que se ocupa de simular el proceso de aprendizaje mediante las redes neuronales artificiales. Estos modelos computacionales se inspiran en el funcionamiento del cerebro humano, en concreto, en las conexiones existentes entre nuestras neuronas, y plantean un mecanismo de aprendizaje por capas, desde lo más concreto hasta lo más abstracto.

Con independencia de la técnica de aprendizaje empleada, lo cierto es que el Machine Learning se fundamenta en el **manejo de datos**. Precisamente, todos los desarrollos en este campo han experimentado un crecimiento vertiginoso en las últimas décadas debido al incremento exponencial de la capacidad de computación, que permite conservar y manejar cantidades desorbitadas de datos. La consultora IDC, citada en la web de TICPymes (2021), estima que en 2025 habremos creado más de 175 zettabytes de datos. ¿Sabes qué es un zettabyte? 10^{21} bytes, es decir, ¡un 1 seguido de 21 ceros!



Qué es BIG DATA y cómo se diferencia de Data Science



El tratamiento de un volumen tan elevado de datos ha propiciado la aparición de un nuevo paradigma, sobre el que seguro que tienes ya un cierto conocimiento: el Big Data. A veces este término se confunde con el de otros que se emplean en este ámbito, como Data Science. Mediante el siguiente vídeo de Rafa Gonzalez Gouveia podrás conocer más sobre ambas disciplinas y, en especial, sobre las principales diferencias existentes. Tras su visualización tendrás que responder tres cuestiones.

Importancia de los datos en Machine Learning

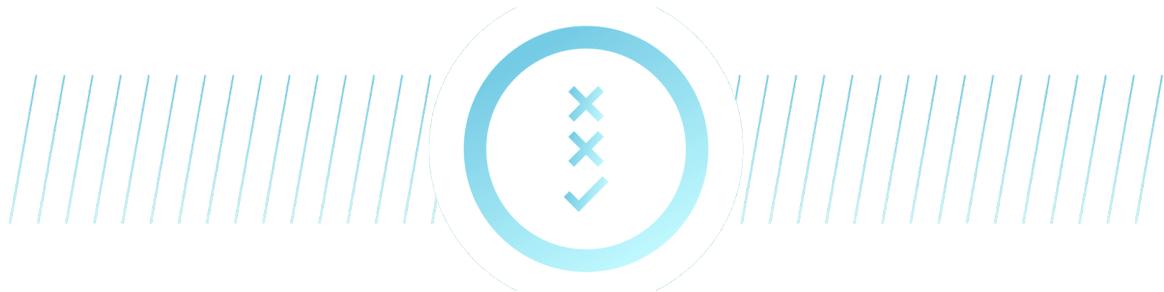
La importancia de los datos en el campo del Machine Learning queda reforzada por el hecho de que la creación y empleo de los modelos que nos permitirán establecer previsiones sobre el futuro se basa en un **aprendizaje extraído de los datos** disponibles. Grus (2019) subraya que un modelo consiste, básicamente, en la especificación de las relaciones que existen entre las distintas variables que intervienen en un problema. Dichas relaciones se pueden concretar bajo **tres enfoques**:

- **Mediante la identificación de las conexiones matemáticas existentes:** el beneficio de una empresa se obtiene como diferencia, es decir, como una resta, entre los ingresos obtenidos en un período y los gastos en los que incurre. Un modelo de negocio tendrá

que considerar tanto los posibles ingresos como los gastos y establecer los cálculos necesarios para determinar las ganancias o pérdidas.



- **A través de procedimientos de prueba y error:** la confección de una receta de cocina implica la determinación de los ingredientes necesarios, junto a la cantidad que se debe emplear, los tiempos y las técnicas de preparación. La mayoría de estas especificaciones proceden de plantear distintas alternativas y seleccionar la que proporcionó un resultado más satisfactorio.



- **Mediante la consideración de relaciones probabilísticas:** los modelos que se emplean en los juegos de azar requieren una valoración de las probabilidades que se tienen en cada momento de ganar al resto de oponentes, todo ello enmarcado en el cumplimiento de sus reglas de funcionamiento.





Existe una rama de las matemáticas denominada Teoría de Juegos que plantea la identificación de las decisiones más acertadas que se pueden tomar en función de las que se encuentran a disposición de otros agentes que tienen capacidad de actuación. Las enseñanzas derivadas de la Teoría de Juegos se aplican en numerosos ámbitos empresariales.

La consideración de los datos para la materialización del aprendizaje y la construcción de modelos constituye un cambio de enfoque muy profundo.

A continuación, conocerás el **enfoque tradicional y los cambios** que ha supuesto la aparición del Machine Learning:

Programación tradicional

Programación tradicional: Lee (2019) señala que en el campo de la **programación tradicional** lo que se hace es introducir los datos (por ejemplo, los ingresos y gastos que se van produciendo), implementar unas rutinas de cálculo (las reglas que permiten elaborar las correspondientes tablas contables) y, con ambos elementos, obtener unos resultados (ganancias o pérdidas experimentadas en el período considerado). El software existente posibilita un tratamiento gráfico de todos estos datos para que la información generada resulte de mayor legibilidad para la persona usuaria. Este funcionamiento tradicional se esquematiza en la siguiente imagen:



Los desarrollos informáticos en dicho contexto lo que persiguen, fundamentalmente, es mejorar la eficiencia de los procesos.

Machine Learning

Machine Learning: los planteamientos en **Machine Learning** han modificado de manera sustancial el anterior paradigma de programación, que ha pasado a ser el que se ilustra en la nueva imagen, también propuesta por Lee (2019):



Esto implica que en el contexto de Machine Learning se elaborará un **algoritmo específico** para cada problemática particular. Como señala Torres (2020), «estos algoritmos aprenden de los datos con el fin de encontrar patrones o tendencias para comprender qué nos dicen estos datos y, de esta manera, construir un modelo para predecir o clasificar los elementos».

Retomando el ejemplo anterior, en este contexto podrías incorporar los datos ya recogidos de ingresos y gastos devengados en períodos previos, que también formaría parte de la salida (ya que a partir de ellos se determinan los resultados obtenidos) y el sistema debería ser capaz de predecir las estimaciones de ganancias o pérdidas futuras o, por ejemplo, determinar, en base a ese histórico, cuáles son los artículos que serán más o menos demandados en los próximos períodos.

Pasamos de programar con el objetivo de alcanzar un objetivo, y dar respuesta a una determinada cuestión, a un nuevo paradigma en el que programas con el objetivo de que el **sistema sea capaz de aprender a conseguir encontrar las respuestas solicitadas**.

Este nuevo enfoque es el que posibilita, según los ejemplos propuestos por Grus (2019), que los sistemas sean capaces de analizar nuevos datos y **responder a preguntas** como las siguientes:

- ¿Este correo que se acaba de recibir es o no *spam*?
- ¿La transacción bancaria que se está procesando es o no fraudulenta?
- ¿Cuál de los mecanismos de propaganda en la web de mis productos resultará más atractivo para futuros adquirientes?
- ¿Qué equipo de fútbol ganará el partido en el que voy a apostar?

Todo ello requerirá un tratamiento en profundidad de datos existentes. Para inferir si un correo electrónico se debe clasificar como spam o no resultará necesario revisar un elevado número de correos para encontrar patrones que permitan hacer una clasificación correcta de los que vayan llegando. ¿Cómo se hará esto? Empleando algún o algunos algoritmos.

Tipos de algoritmos en Machine Learning

Teniendo en mente la idea de que para deducir cómo clasificar un correo electrónico habrá que revisar una gran cantidad de ellos para encontrar patrones, debes saber que los **algoritmos** que se emplean en Machine Learning se clasifican en **dos grandes grupos**:

- **Algoritmos de aprendizaje supervisado:** en este contexto los datos empleados para que el sistema aprenda (mediante su entrenamiento) incluyen la solución deseada, que recibe el nombre de **etiqueta** (o *label*, que es un término que se emplea de manera habitual). El aprendizaje persigue la construcción de un modelo cuya aplicación debe permitir la **correcta identificación de los datos de entrada** que se presenten por primera vez.

En el caso de la clasificación de los correos, durante la fase de entrenamiento se presentará al sistema un conjunto de *e-mails*, clasificados como spam y como legítimos. Como señala Torres (2020) la construcción del modelo se realizará de manera iterativa, comparando los resultados de sus predicciones con la respuesta correcta, lo que permitirá ir afinando el comportamiento. En el blog de Bismart (s.f.) se recogen las principales **características** de dichos algoritmos:

- Requieren la intervención humana en el proceso de **etiquetado** de los datos y para introducirlos en las rutinas de entrenamiento.
- El modelo proporcionará una **salida esperada**, en el sentido de que hemos identificado previamente las categorías en las que se pueden encuadrar los datos.
- Se suelen emplear para alcanzar alguna de las **dos siguientes metas**:
 - La realización de **clasificaciones** (un mensaje es o no spam, la transacción bancaria es legal o fraudulenta, etc.).

- La **predicción** de valores futuros, en base a lo que se conoce como regresión (cuánto venderemos de un producto, qué beneficios alcanzará la empresa en el próximo trimestre, etc.).
- **Algoritmos de aprendizaje no supervisado:** los sistemas aprenden empleando **datos no etiquetados**. El objetivo del algoritmo será proceder a una **clasificación en base a posibles relaciones internas**. En el blog de Bismart (s.f.) se recogen también las principales **características** de los algoritmos:
 - La intervención humana se centra en la **identificación de los objetivos** que se desean alcanzar, aunque las personas no actuarán directamente, como en el caso anterior, en los bucles de aprendizaje.
 - Se pueden distinguir **dos tipos de técnicas** para obtener clasificaciones consistentes:
 - **Clustering:** proporciona distintos agrupamientos de los datos. El sistema, por ejemplo, puede distinguir diferentes tipos de clientela en base a algún parámetro que desconocemos y que el modelo conseguirá identificar.
 - **Asociación:** permite identificar relaciones ocultas en los datos, por ejemplo, tipologías de clientela que, si adquieren un producto, seguramente estarán interesadas en otro de naturaleza complementaria.

Grus (2019) describe **otros planteamientos**, el primero de los cuales es híbrido entre los dos descritos, como:

- **Algoritmos semisupervisados:** en los que se etiquetan algunos datos, pero no todos.
- **Algoritmos on-line:** que requieren un ajuste continuo del modelo en función de los nuevos datos recibidos. Normalmente se implementan en contextos web, aunque también tienen un importante campo de aplicación en sistemas de control en tiempo real.
- **Algoritmos de refuerzo:** en los que, tras la realización de una serie de predicciones, el modelo recibe *feedback* sobre el nivel de los resultados obtenidos.

Recapitulación parcial

Has empezado por definir en qué consiste la inteligencia artificial, enfatizando que persigue la imitación de los comportamientos de los seres humanos, y los tipos en los que se puede clasificar.

También has descrito el Machine Learning como un subconjunto de la inteligencia artificial que se centra en la reproducción de cómo aprenden los seres humanos.

Además, has identificado el enfoque tradicional y los cambios que ha supuesto la aparición del Machine Learning, pasando a un nuevo paradigma en el que se programa para que el sistema sea capaz de aprender a conseguir encontrar las respuestas solicitadas. Para programar, has descubierto que los algoritmos que se emplean en Machine Learning se clasifican en dos grandes grupos: de aprendizaje supervisado y no supervisado.



RECAPITULACIÓN

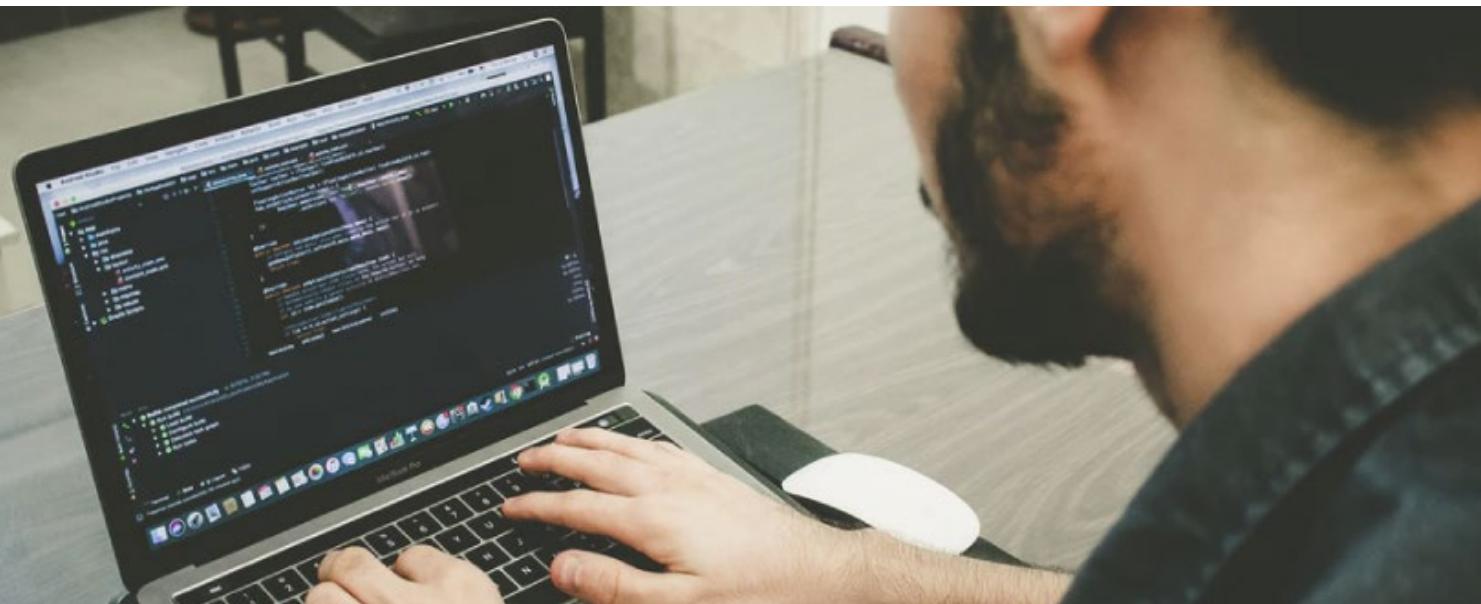
Has comenzado con la identificación de los lenguajes de programación con códigos que permiten la comunicación con los dispositivos y la transmisión de las órdenes pertinentes. Los lenguajes más empleados son los de alto nivel, que facilitan la generación de un código legible, dado su elevado nivel de abstracción. Estos lenguajes, a su vez, pueden ser interpretados o compilados.

Python es uno de los lenguajes de programación más populares en la actualidad. Se caracteriza por ser interpretado, de código abierto y multiparadigma, resultando uno de los más sencillos de aprender. Posee un gran potencial para los desarrollos en el área de Machine Learning. Es fundamental manejarlo para abordar tareas de programación en dicho ámbito.

También has podido comprobar la importancia del empleo de librerías en Python, que dispone de una librería estándar, con numerosos elementos. Los más valiosos para trabajar en el ámbito de Machine Learning son las funciones Built-in o incorporadas y los módulos numéricos y matemáticos. Las librerías desarrolladas por personas especializadas reciben el nombre de externas y las más relevantes se disponen en el Python Package Index.

Además, has podido introducirte en el estudio del Machine Learning. La inteligencia artificial persigue la imitación de los modelos de comportamiento humano en el contexto de las máquinas y de los dispositivos informáticos. Las técnicas desarrolladas han logrado que los sistemas artificiales realicen tareas muy específicas de manera eficiente, aunque la implementación del control de numerosas tareas, diferentes entre sí, todavía se percibe como muy lejano.

Las técnicas de Machine Learning constituyen una parte de las que integran la inteligencia artificial, aunque tienen importancia nuclear, al enfocarse en la reproducción de los procedimientos de aprendizaje de los seres humanos. Fundamentalmente, persigue que los sistemas informáticos sean capaces de aprender en base a datos existentes, para lo que se pueden emplear dos destacados enfoques, el aprendizaje supervisado y el no supervisado.



BIBLIOGRAFÍA

- Aprende Web. (s.f.). *Librerías. Facilitar la programación.* Consultado el 03 de marzo de 2022.
<https://aprende-web.net/librerias/>
- Bismart. (s.f.). *Diferencias entre el Machine Learning supervisado y no supervisado.* Consultado el 03 de marzo de 2022.
<https://blog.bismart.com/es/diferencias-machine-learning-supervisado-no-supervisado>
- César, J. (13 de septiembre de 2020). *Las 10 principales empresas de tecnología que utilizan Python en sus productos.* Facialix.
<https://blog.facialix.com/las-10-principales-empresas-de-tecnologia-que-utilizan-python-en-sus-productos/>
- Chakray. (s.f.). *Lenguajes de programación: tipos y características.* Consultado el 03 de marzo de 2022.
<https://www.chakray.com/es/lenguajes-programacion-tipos-caracteristicas/>
- Craft Code. (s.f.). *La guía de las buenas prácticas en programación.* Consultado el 03 de marzo de 2022.
<https://craft-code.com/las-buenas-practicas-en-programacion/>
- García, P., Salas, P., Gutiérrez, D., González, I. y Durán, M. J. (2019). *Aprendiendo a programar en Python.* Universidad de Málaga.
- GNU. (s.f.). *¿Qué es el software libre?* Consultado el 03 de marzo de 2022.
<https://www.gnu.org/philosophy/free-sw.es.html>
- Grus, J. (2019). *Data Science from Scratch. First Principles with Python.* Second Edition. O'Reilly Media Inc. Sebastopol.

Herranz, A. (27 de abril de 2021). *Estas son las comunidades de desarrollo más grandes y las que más crecen en los últimos meses*. Xataka.

<https://www.xataka.com/pro/estas-comunidades-desarrollo-grandes-que-crecen-ultimos-meses>

- Hola mundo. (s.f.). COBOL. Consultado el 03 de marzo de 2022.
<http://www.holamundo.es/lenguaje/cobol/>
- Kumar, C. (10 de noviembre de 2020). *12 mejores IDE que todo programador debe conocer*. Geekflare.
<https://geekflare.com/es/ide-for-programmer/>
- Laura M. (19 de diciembre de 2021). *¿Cuál es el mejor lenguaje de programación del 2021?* BitDegree.
<https://es.bitdegree.org/tutoriales/mejor-lenguaje-de-programacion/>
- Lee, W. (2019). *Python Machine Learning*. Ed. John Wiley & Sons, Incorporated.
- Lenguajes de programación. (s.f.). *Hola mundo*. Consultado el 03 de marzo de 2022.
<https://lenguajesdeprogramacion.net/hola-mundo/>
- Lenguajes de programación. (s.f.). *¿Qué es un lenguaje interpretado?* Consultado el 03 de marzo de 2022.
<https://lenguajesdeprogramacion.net/diccionario/que-es-un-lenguaje-interpretado/>
- López, M. (16 de junio de 2020). *Qué es un lenguaje de programación*. OpenWebinars.
<https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion/>
- Martínez, M. (30 de abril de 2021). *Los 10 mejores editores de texto para desarrolladores y diseñadores*. Profile.
<https://profile.es/blog/mejores-editores-texto/>
- Martínez, M. (09 de junio de 2020). *¿Qué son los paradigmas de programación?* Profile.
<https://profile.es/blog/que-son-los-paradigmas-de-programacion/>
- Martínez, J. (10 de octubre de 2020). *15 librerías de Python para Machine Learning*. IArtificial.net.
<https://www.iartificial.net/librerias-de-python-para-machine-learning/#Seaborn>
- Matthes, E. (2021). *Curso intensivo de Python*. 2.ª edición. Anaya Multimedia. Madrid.
- MCLIBRE. (s.f.). Historia de Python. Consultado el 03 de marzo de 2022.
<https://www.mclibre.org/consultar/python/otros/historia.html>
- Microsoft. (23 de septiembre de 2021). *Introducción al uso de Python en Windows para principiantes*.
<https://docs.microsoft.com/es-es/windows/python/beginners>
- Python. (s.f.). *Las Funciones Built-in*. Consultado el 03 de marzo de 2022.
<https://docs.python.org/es/3/library/functions.html>
- El Pythonista. (s.f.). *¿Qué es Python?* Consultado el 03 de marzo de 2022.
<https://elpythonista.com/python>

- Pulido, M. (23 de septiembre de 2019). *Todo lo que deberías saber sobre programación reactiva*. Slashmobility.
<https://slashmobility.com/blog/2019/09/programacion-reactiva/>
<https://empresas.blogthinkbig.com/las-5-razones-por-las-que-todo-el-mundo-quiere-aprender-python/>
- Real Academia Española. (s.f.). Aprender. En *Diccionario de la lengua española*. Consultado el 03 de marzo de 2022.
<https://dle.rae.es/aprender?m=form>
- Real Academia Española. (s.f.). Inteligencia. En *Diccionario de la lengua española*. Consultado el 03 de marzo de 2022.
<https://dle.rae.es/inteligencia?m=form>
- Real Academia Española. (s.f.). Inteligencia artificial. En *Diccionario de la lengua española*. Consultado el 03 de marzo de 2022.
<https://dle.rae.es/inteligencia?m=form>
- Recuero, P. (05 de febrero de 2020). *Las 5 razones por las que todo el mundo quiere aprender Python*. Telefónica Tech.
- Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach (fourth edition)*. Pearson.
- Santander Universidad (8 de marzo de 2021). *¿Qué son los lenguajes de programación y cuáles son los más utilizados?*
<https://www.becas-santander.com/es/blog/lenguaje-programacion.html>
- Significados. (s.f.). *Significado de Lenguaje*. Consultado el 03 de marzo de 2022.
<https://www.significados.com/lenguaje/>
- Solver. (19 de junio de 2021). *6 librerías imprescindibles de Python para Machine Learning*.
<https://iasolver.es/6-librerias-de-python-para-machine-learning/>
- tiThink. (13 de junio de 2018). *12 buenas prácticas para el desarrollo software*.
<https://www.tithink.com/es/2018/06/13/12-buenas-practicas-para-el-desarrollo-software/>
- TICPyMes. (04 de noviembre de 2021). *La gestión del dato en las empresas: ¿Cómo pueden sacar el mayor partido de ellos?* Consultado el 03 de marzo de 2022.
<https://www.ticpymes.es/tecnologia/noticias/1129328049504/gestion-del-dato-empresas-pueden-sacar-mayor-partido-de.1.html>
- Tokio School. (22 de noviembre de 2020). *La historia de Python. Las versiones de un lenguaje único*.
<https://www.tokioschool.com/noticias/historia-python/>
- Torres, J. (2020). *Python Deep Learning*. Marcombo, S.L. España.
- Yáñez, C. (12 de marzo de 2021). *¿Qué es un lenguaje de programación y qué tipos existen?* DEUSTO Formación.
<https://www.deustoformacion.com/blog/programacion-tic/que-es-lenguaje-programacion-que-tipos-existen>



DESCARGABLES

- Sergio Delgado Quintero. *Aprende Python.*
https://aprendepython.es/_downloads/907b5202c1466977a8d6bd3a2641453f/aprendepython.pdf
- Alfredo Sánchez Alberca. *Referencias.*
<https://aprendeconalf.es/docencia/python/manual/manual-python.pdf>.

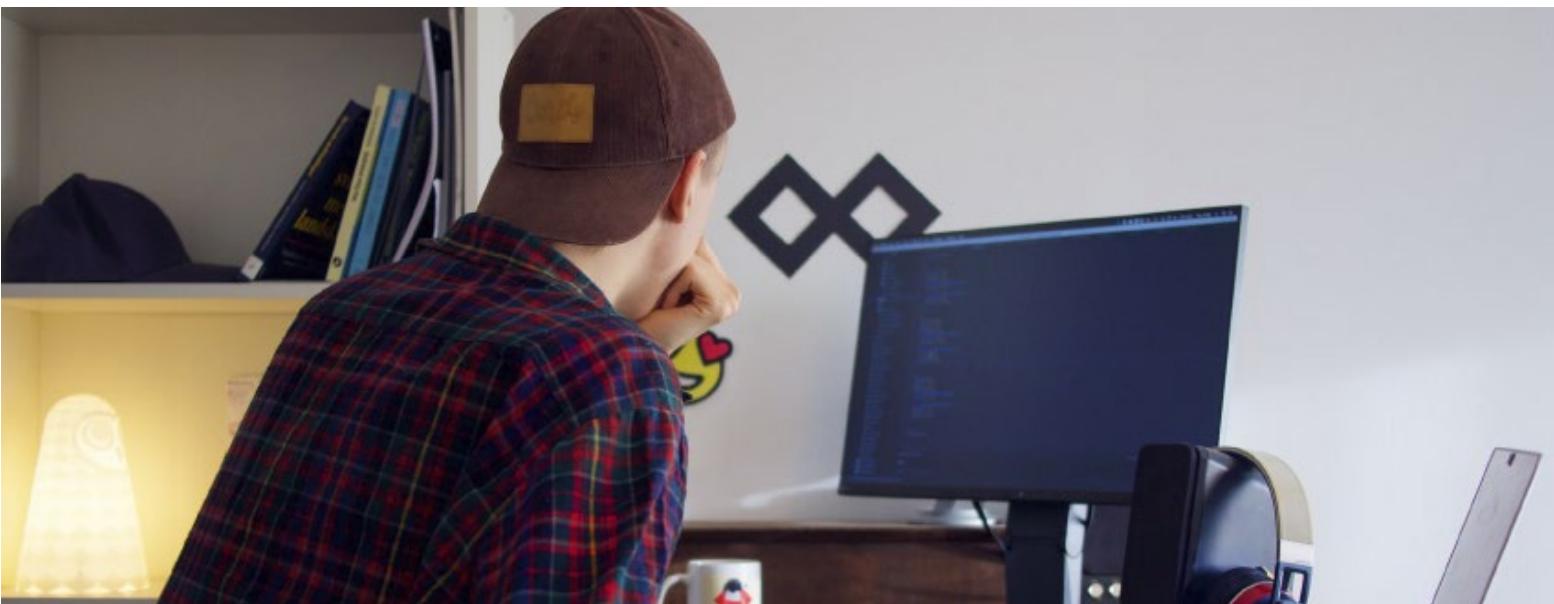


ENLACES DE INTERÉS

- Computer hoy. 8 lenguajes de programación para aprender a programar desde cero.
<https://computerhoy.com/listas/industria/lenguajes-programacion-aprender-desde-cero-617085>
- Xataka. 17 desarrolladores nos cuentan qué lenguaje de programación elegirían para empezar desde cero y por qué.
<https://www.xataka.com/especiales/16-desarrolladores-nos-cuentan-que-lenguaje-programacion-elegirian-para-empezar-cero-que>
- Stack Overflow en español. Foro sobre programación.
<https://es.stackoverflow.com/>
- Python. Página web de Python.
<https://www.python.org/>
- Visual Studio Code. Página web de Visual Studio Code.
<https://code.visualstudio.com/>
- Python Package Index. Web de Python Package Index.
<https://pypi.org/>
- Delft Stack. Deshabilitar el límite de longitud de la ruta en Python.
<https://www.delftstack.com/es/howto/python/disable-path-length-limit-python/>
- Python. La Biblioteca Estándar de Python.
<https://docs.python.org/es/3/library/index.html>
- Tecnologías Información. Sistemas Expertos: definición, aplicaciones y ejemplos.
<https://www.tecnologias-informacion.com/sistemas-expertos.html>

Fuentes de vídeo

- Cuatro principios de la programación orientada a objetos.
<https://www.youtube.com/watch?v=tTPeP5dVuA4>
- Funcionamiento de una librería Python.
<https://vimeo.com/694375721>
- ¿Qué es la inteligencia artificial?
<https://www.youtube.com/watch?v=NSf3o-wxtQ0>
- Qué es BIG DATA y cómo se diferencia de Data Science.
<https://www.youtube.com/watch?v=NKWjXoO3a7k>



PREGUNTAS FRECUENTES

- **¿Qué es una máquina virtual?**

Se trata de un software especial, que permite simular en tu equipo real el funcionamiento de otro ordenador virtual, que dispondría de los correspondientes elementos de hardware. De este modo, resultará posible ejecutar en dicho ordenador un sistema operativo diferente o una aplicación con condicionantes especiales, de una manera absolutamente transparente para la persona usuaria. Todo este comportamiento se encapsula en una ventana de trabajo, de forma independiente de otras tareas del equipo real.

- **¿Qué es un framework?**

Es un marco de trabajo, integrado por herramientas de programación y módulos, que facilita la tarea de desarrollo de aplicaciones. Posibilita, por ejemplo, la disposición de código repetitivo en una determinada ubicación, permite trabajar de manera colaborativa, resulta más segura y, por lo común, suele contar con una comunidad de personas usuarias que lo emplean, lo que facilita la resolución de las dudas de empleo que pudieran surgir.

- **¿Qué significa DevOps?**

Este término se obtiene como combinación de dos términos en inglés: *development* (desarrollo) y *operations* (operaciones). Por tanto, hace referencia a un planteamiento de trabajo en ámbitos tecnológicos que persigue la articulación de las personas y los procesos para ofrecer productos de mayor calidad y mejor rendimiento.

- **¿Qué es el lenguaje SQL?**

El lenguaje de consultas estructuradas (las siglas SQL significan *Structured Query Language*) se emplea fundamentalmente para la administración de bases de datos relacionales, es decir, caracterizadas por una estructura que facilita el almacenamiento y la gestión de dichos datos. Desde la década de los 80, SQL constituye el lenguaje estándar para esta tipología de bases de datos.

- **¿Qué es un algoritmo?**

En el ámbito de la informática, un algoritmo es un conjunto de instrucciones ordenadas que permiten resolver un problema o llevar a cabo una determinada tarea. La plasmación de uno o distintos algoritmos en un lenguaje de programación es lo que recibe el nombre de programa informático.



GLOSARIO

- **A**
 - **Aleatorio**

Se dice de algo que depende del azar o de la suerte y que, por tanto, no se puede determinar previamente.
 - **Alfanumérico**

Elemento que está integrado tanto por números como por letras, de manera conjunta.
 - **Asíncrono**

Es la característica de hechos relacionados que no se producen de manera coordinada en el tiempo, sino cada uno en un instante diferente. Por ejemplo, una conversación telefónica es, por naturaleza, un evento síncrono, dado que las personas intervenientes han de estar conectadas a la vez. Por el contrario, la mensajería electrónica es asíncrona, dado que un individuo puede escribir en un momento dado y la persona receptora leer el mensaje en otro instante y contestarlo en su caso cuando desee.
- **B**
 - **Bucle**

Secuencia de instrucciones en un programa que se repite numerosas veces hasta que se cumple alguna condición que hace que se detenga.
- **C**
 - **Criptografía**

Ciencia especializada en la escritura de contenidos mediante procedimientos secretos, de modo que resulten ininteligibles salvo para las personas que dispongan de la clave o el conocimiento adecuado.

- **F**
 - **Feedback**

Retorno que proporciona un sistema y que sirve, por lo general, para comparar la respuesta obtenida con la esperada.
- **L**
 - **Licencia**

Tipología de contrato que se establece en el ámbito informático, por el que la persona que desarrolla un software concede determinados permisos a terceros para que puedan emplearlo.
- **R**
 - **Repositorio**

Depósito virtual en el que se almacenan, se organizan y se mantienen archivos digitales.
- **V**
 - **Virtualidad**

Cualidad de algo que no es real ni tangible.



IDEASPROPIAS
editorial