# **Módulo 0**

**Básicos**

* Muy breve historia de la computadora
* Personal computer
* Lenguajes de programación
* Lenguajes compilados e interpretados.

### Lenguajes de alto y bajo nivel.

* Herramientas de desarrollo
* Preparados, listos, a instalar!!!!!!!!!!
* Entornos interactivos, guiones y **'Hola Mundo'**.

**Manejo de IDE Jupyter**

* Ejecución en Jupyter.
* El manejo de celdas
* Markdown y HTML
* Markdown
* Etiquetas HTML

**Micro historia de** [**Python**](http://python.org/)

**Retro - compatibilidad negativa**

# **Minería de datos (data mining)**

La minería de datos o exploración de datos es una etapa de análisis de "knowledge discovery in databases" o KDD. Es un campo de la estadística y las ciencias de la computación referido al proceso de análisis cuidadoso que intenta descubrir patrones en grandes volúmenes o conjuntos de datos. El fin es encontrar en el pajar la información relevante que se pueda usar con un propósito específico, la aguja .

La minería de datos es parte fundamental para la ciencia de los datos y la inteligencia comercial y se centra buscar en patrones y relaciones dentro de conjuntos de datos.

* Extraer información previamente recolectada (en bruto).
* Transformarla en una estructura comprensible para su uso posterior.
* Etapa de análisis en bruto, gestión de datos, procesamiento de datos, de modelo y de las consideraciones de inferencia, de métricas de intereses, de consideraciones de la teoría de la complejidad computacional, de post-procesamiento de las estructuras descubiertas, de la visualización y de la actualización en línea.

Ejemplos de estos son el extraer patrones interesantes hasta ahora desconocidos, como los grupos de registros de datos (análisis clúster), registros poco usuales (la detección de anomalías), dependencias (minería por reglas de asociación), entre otros.

* Modelos estadísticos como bases de informes de resultados
* Ten en cuenta que ni la recolección o preparación de datos, ni por otro lado la interpretación de los resultados y la su información son parte de la etapa de minería de datos. Pero que también son procesos de KDD como pasos adicionales.

Un proyecto de minería de datos tiene cinco fases necesarias que son, esencialmente:

* Comprensión: del negocio y del problema que se quiere resolver.
* Determinación, obtención y limpieza: de los datos necesarios.
* Creación de modelos matemáticos.
* Validación, comunicación: de los resultados obtenidos.
* Integración: si procede, de los resultados en un sistema transaccional o similar.

Fuente [https://es.wikipedia.org/wiki/Miner%C3%ADa\_de\_datos](https://es.wikipedia.org/wiki/Minería_de_datos)

**Se recomienda la lectura total de este artículo**

**Muy breve historia de la computadora**

En 1840 Charles Babbage formalmente inicio la informática con una máquina de calcular programable con un Motor Analítico basado en **información manipulada mediante tarjetas perforadas**.

La matemática [Ada Lovelace](https://en.wikipedia.org/wiki/Ada_Lovelace), con solo veinticinco años, colaboró con Babbage y en 1840 tuvo un papel fundamental en la divulgación de la máquina de calcular, de su sistema de programación y fue la primera que entendió sus posibilidades más allá del cálculo de fórmulas.

|  |  |
| --- | --- |
| Averiguar :  1) El entorno y que conocimientos había en la época  2) ¿Como obtuvo su conocimiento matemático?  3) Situarse en la época, sexo, contexto social y tratar de definir como las ideas innovadoras de Ada tuvieron acogida entre sus colegas. |  |

En 1843 publicó el trabajo "Sketch of the analytical engine invented by Charles Babbage" en el que describe el Motor Analítico, añade reflexiones propias sobre el alcance del invento y construye un ejemplo completo, con tablas y diagramas, de cómo hacer que la máquina produzca la secuencia de los números de Bernoulli. **Estas tablas y diagramas se reconocen como el **primer programa de un computador**.**

En 1936, [Alan Turing](https://en.wikipedia.org/wiki/Alan_Turing) creo la idea de un computador, utilizando un modelo muy sencillo de procesamiento que **l**e**ería y escribiría ceros y unos** en un soporte informático de proceso -memoria provisto una cinta y un programa escrito en función de una tabla definida en la máquina.

|  |  |
| --- | --- |
|  |  |

·

|  |  |
| --- | --- |
|  | Averiguar :  1) El entorno y que conocimientos habia en la época  2) Que empresa de primer nivel rinde homenaje con su logo y porque  3) Tras fu fallecimiento basada en una causa judicial que lo castigaría por un delito de la época. El país y sistema judicial que lo persiguió, tomo acciones que a la comunidad indigno sobremanera.  A) Pidieron disculpas y borraron del expediente el supuesto delito.  B) Reconocieron el error explicando que en esa época era una convención.  C) Se acepto la culpabilidad en la causa pero se indulto, perdón real, póstumamente.   ¿Cual de los puntos es parte de nuestra realidad, en que año fue y cuanto paso tras la muerte de A.Turing? |

El genio del matemático inglés solo mediante la teoría abstracta, ya que en esos tiempos aun no existía en la realidad, modela la [Máquina de Turing](https://en.wikipedia.org/wiki/Turing_machine) y demuestra que sí un problema es o no computable y **establece con su trabajo los límites de la computación**. Se mostraba que es posible a una máquina escribir información en una cinta que tomen como datos de origen otra información de otra(s) cintas(s) según procedimientos establecidos en la maquina (programa).

En medio de un clima de enormes avances entre la primera y segunda guerra mundial hubo una evolución en electrónica y electromecánica que dio base a la computación.

El empuje de la segunda guerra donde ya se necesitaba estructuras lógicas de manipulación de información, logística, disposición de fuerzas aliadas y enemigas, movilidad terrestre, fluvial y aérea, transmisión y acumulación de informes de las cadenas de personal, radares, etc. Se necesitava evaluar datos y mas datos en muy poco tiempo ya que las circunstancias ya no iban a caballo sino que cambiaban a la velocidad de la luz. Un ejemplo de esto es el inicio de la criptografía en la comunicación – Enigma.

Max Newman y Alan Turing trabajaron juntos durante la Segunda Guerra Mundial en Bletchley Park, el centro de decodificación británico encargado de romper los códigos alemanes, incluido el cifrado de la máquina Enigma. Allí, contribuyó al desarrollo y uso de la máquina "Colossus", la primera computadora digital electrónica programable, diseñada para descifrar mensajes cifrados por la máquina alemana "Lorenz SZ 40/42".

|  |  |  |
| --- | --- | --- |
|  | | Averiguar la velocidad de computo, la memoria que poseía, tamaño, peso y precio. Compararlo con el celular de modelo mas económico (a su elección) del mercado en este momento |
|  | Con [ENIAC](https://en.wikipedia.org/wiki/ENIAC) (Electronic Numerical Integrator and Computer), (1945) se creo la base de la arquitectura de almacenamiento vigente a la fecha. [John Von Neumann](https://en.wikipedia.org/wiki/John_von_Neumann), proponen las dos ideas claves, el programa almacenado en memoria y un conjunto de instrucciones de procesamiento que incluye el direccionamiento indirecto. | |
|  | |

|  |
| --- |
| En 1948 F.C. Williams y Tom Kilbur con la colaboración de Geoff Tootill y Max Newmann dan a luz a [Baby](https://en.wikipedia.org/wiki/Manchester_Small-Scale_Experimental_Machine), (también conocido como "Manchester Baby" o "Manchester Mark I") con la nueva memoria electrónica a válvulas (e tubos de vacío ) en y para la universidad de Manchester Baby fue el primer computador con un conjunto de instrucciones completo, capaz de realizar saltos, condicionales y direccionamiento indirecto. Demostrar el funcionamiento de la lógica electrónica y **nació la memoria de acceso aleatorio (RAM).** |

El primer **computador comercial** fue el UNIVAC I (Universal Automatic Computer) fue diseñado por J. Presper Eckert y John Mauchly y construido por la empresa Remington Rand en los Estados Unidos. El primero se instala en 1951 para Departamento de Censo de los EEUU.

Con la colaboración de Turing en 1952 Christopher Strachey programa un jugador de damas fue el primer programa complejo de Inteligencia Artificial se ejecutó en el verano de en el Ferranti Mark I en el Laboratorio de Computación de Manchester.

Turing participó también en el desarrollo de otros programas de IA, como un jugador de ajedrez basado en heurísticas.

El IBM 704 fue el otro gran ordenador comercial de la década de los 50 con una comercialización y difusión mucho mayor que el UNIVAC

El UNIVAC I era una máquina interesante para programar, con su almacenamiento basado en líneas de retardo de mercurio y su propensión a fallar. Los programas se introducían en el computador tecleándo los en cintas magnéticas, una innovación importante en ese tiempo.

El trabajo con el IBM 704 en la Universidad de NY fue una experiencia radicalmente distinta de la del UNIVAC I. Fue construido para ejecutar aplicaciones científicas, y su principal innovación era una memoria de núcleo magnético, reemplazando la memoria de tubos Williams del IBM 701. **También tenía una unidad aritmética en punto flotante. La máquina tenía el equivalente a 128 KB de memoria principal, 32 KB de memoria secundaria y cintas magnéticas que podía almacenar 5 MB de datos. Operaba a 0.04 MIPS y costó 3 millones de dólares en 1957.**

**Por favor relacione la memoria de trabajo con la de su pc y celular**

**Personal computer:**

El Intel 8080 fue uno de los primeros microprocesador comerciales lanzado al mercado por Intel Corporation en abril de 1974 con 8 bits y una velocidad de reloj de 2 MHz. Este procesador fue utilizado en la primera computadora personal de éxito masivo: el Altair 8800 lanzada en 1975

El microprocesador Zilog Z80A de 8 bits aa3.5 MHz dio vida a las populares Sinclair ZX Spectrum de Sinclair Research Ltd en la década de 1980. Luego se mejoro el micro incluyendo 64 KB de RAM, mayor capacidad de almacenamiento y mejoras en el color y el sonido.

**Arquitectura x86**

**Intel 8086:** Lanzado en 1978, el 8086 fue el primer microprocesador de **16 bits** de Intel. Su arquitectura de 16 bits marcó el inicio de la serie x86. Aunque fue diseñado inicialmente para aplicaciones embebidas e industriales, la compatibilidad con el anterior Intel **8080 de 8 bits** lo hizo popular en el mercado de computadoras personales. El 8086 y su sucesor, el **8088 (que tenía un bus de datos de 8 bits**), fueron utilizados en las primeras computadoras IBM PC y clonadas.

**Intel 80286 (o i286):** Lanzado en 1982, el 80286 fue un gran avance con una arquitectura de 16 bits mejorada, con capacidad para ejecutar software de modo protegido y modo real, una mayor seguridad y multitarea y un gran aumentó en velocidad de procesamiento.

**Intel 80386 (o i386):** Lanzado en 1985, introdujo la arquitectura de 32 bits y fue el primer procesador de Intel en ofrecer la capacidad de acceso a memoria virtual, lo que permitía a los sistemas operativos modernos utilizar la memoria de manera más eficiente y admitir multitarea real y protegida. El 80386 también fue el primero en introducir la arquitectura de conjunto de instrucciones (ISA) x86 de 32 bits.

**Intel 80486 (o i486):** Lanzado en 1989, el 80486 fue una mejora respecto al 80386. Ofrecía una mayor velocidad de procesamiento y mejoras en la caché.

Hubo mas de una gama (486 DLC) e I486 al que se le podía colocar un co-procesador matemático

**Intel Pentium (80586):** Lanzado en 1993 bajo la nomenclatura o nombre "Pentium" fue usado por Intel para la quinta generación de procesadores x86.

**AMD de 64 bits, (AMD64 o x86-64):** Lanzado en 2003 con su procesador AMD Athlon 64. Esta arquitectura extendió la arquitectura de 32 bits x86 existente para soportar registros de 64 bits y direcciones de memoria de 64 bits, lo que permite a los procesadores AMD64 manejar cantidades de memoria y cálculos más grandes. Los procesadores AMD64 son capaces de ejecutar aplicaciones de 32 bits y sistemas operativos de manera nativa, lo que facilitó la transición a procesadores de 64 bits para los usuarios y desarrolladores.

**Arquitectura NO x86**

**ARM (Advanced RISC Machine):** ARM es una arquitectura de procesadores RISC (Reduced Instruction Set Computing) que se ha vuelto muy popular en dispositivos móviles, como teléfonos inteligentes y tabletas. Además de su presencia en dispositivos móviles, ARM también ha ganado tracción en servidores y sistemas embebidos debido a su eficiencia energética y rendimiento.

**RISC-V: RISC-V** es una arquitectura de procesadores de código abierto basada en el conjunto de instrucciones RISC. Se ha vuelto cada vez más popular debido a su naturaleza abierta y su flexibilidad, lo que permite a los diseñadores de chips personalizar la arquitectura para sus necesidades específicas. RISC-V se ha utilizado en una variedad de aplicaciones, desde sistemas embebidos hasta supercomputadoras.

**IBM Power:** Es otra alternativa a x86 que se ha utilizado en servidores y sistemas de alto rendimiento. Los procesadores Power se han destacado por su escalabilidad y capacidad para manejar cargas de trabajo intensivas.

**SPARC(Scalable Processor Architecture): E**s una arquitectura de procesadores desarrollada por Oracle Corporation. Aunque su presencia ha disminuido en los últimos años, SPARC ha sido utilizado en sistemas de alto rendimiento y servidores empresariales.

**Intel Itanium:** Aunque no ha logrado el mismo nivel de éxito que x86, Intel Itanium es una arquitectura que ha sido utilizada en servidores y sistemas de misión crítica. Está diseñada para manejar grandes cargas de trabajo y paralelismo.

**Computación cuántica:**

Es un campo emergente de la informática que se basa en los principios de la mecánica cuántica para procesar información y realizar cálculos.

A diferencia de la computación clásica, que utiliza bits binarios (0 o 1) para almacenar y procesar información, la computación cuántica utiliza "qubits", que son unidades de información cuántica que pueden existir en múltiples estados simultáneamente debido al fenómeno conocido como superposición cuántica.

La superposición cuántica permite que los qubits realicen cálculos de manera exponencialmente más rápida que los bits clásicos, lo que tiene el potencial de resolver problemas que serían prácticamente imposibles o requerirían mucho tiempo con la computación clásica. Además, otra propiedad fundamental de la computación cuántica es la "entrelazación cuántica", que permite que los qubits estén intrínsecamente conectados entre sí y compartan información instantáneamente, sin importar la distancia que los separe, lo que ofrece una ventaja potencial para ciertas aplicaciones de comunicación y criptografía.

La computación cuántica todavía está en sus primeras etapas y enfrenta muchos desafíos técnicos. Actualmente, los científicos e ingenieros están trabajando en el desarrollo de hardware cuántico, como qubits más estables y sistemas de corrección de errores cuánticos, para superar las dificultades asociadas con la decoherencia cuántica y los errores cuánticos.

Se espera que en el futuro, la computación cuántica tenga un impacto significativo en diversas áreas, como la optimización de procesos, la simulación de sistemas complejos, el descifrado de claves criptográficas y el avance en la inteligencia artificial, entre otras aplicaciones.

**Lenguajes de programación:**

En 1940 con el desarrollo de las primeras computadoras electrónicas se crea lenguaje ensamblador ("assembly") para proporcionar un nivel más cercano al lenguaje máquina, pero al mismo tiempo más fácil de entender y programar para los seres humanos.

ENIAC fue una de las primeras computadoras en utilizar un lenguaje ensamblador. Sin embargo, el primer lenguaje ensamblador moderno fue desarrollado para la computadora UNIVAC I en 1951.

Ya se permitía a los programadores escribir instrucciones usando mnemónicos (abreviaturas mnemotécnicas) para representar las operaciones que se ejecutaban en la computadora.

La evolucionando para adaptarse a diferentes arquitecturas de computadoras ( hardware) y nuevas orientaciones y requerimientos en la programación (software) con tiempo llevó a la creación de compiladores que traducían estos lenguajes de alto nivel a lenguaje ensamblador o lenguaje máquina, lo que facilitó aún más la programación de computadoras.

John W. Backus e IBM (International Business Machines Corporation) lanza oficialmente el 20 de octubre de 1956 "**FORTRAN**", ("**FOR**mula **TRAN**slator" traductor de fórmulas), es uno de los primeros lenguajes de programación de alto nivel y fue diseñado específicamente para aplicaciones científicas y de ingeniería.

• El primer manual de FORTRAN se imprime en octubre de 1956 para el IBM 704.

• El primer compilador se comercializa en abril de 1956.

**C:** Dennis Ritchie en los laboratorios Bell (Bell Labs) lanza en la década de 1970 **"C"** un lenguaje de programación de alto nivel, se creado originalmente para implementar el sistema operativo Unix (tambien de Bell Labs). Hoy "C" sigue siendo uno de los lenguajes de programación más populares y ampliamente utilizados en la industria del software pero orientado a propósito general que ofrece un buen equilibrio entre la programación de alto nivel y el acceso de bajo nivel a la memoria y los recursos del sistema. Es una elección ideal para desarrollar aplicaciones de sistema, controladores de hardware, software embebido por su eficiencia y su capacidad para acceder directamente a la memoria y manipular datos a nivel de bits. Los programas escritos en "C" pueden ser fácilmente trasladados - portados entre diferentes plataformas y sistemas operativos .

Los primeros lenguajes de alto nivel se desarrollaron a finales de la década de los 50:

• FORTRAN en 1956

• Lisp en 1958

Ambos lenguajes planteaban dos enfoques muy distintos desde el principio:

• FORTRAN

◦ Primer lenguaje comercial, equipo de IBM dirigido por John W. Backus

◦ Lenguaje imperativo: estado, estructuras de control, contador de programa, celdas de memoria

 ◦ Lenguaje compilado

• Lisp

◦ Lenguaje diseñado en un departamento de investigación, un equipo del MIT dirigido por John McCarthy

◦ Lenguaje funcional: funciones, recursión, listas, símbolos

◦ Lenguaje interpretado

Un lenguaje es un conjunto de cadenas de símbolos con los que se pueden crear mensajes. De ese modo los mensajes son transmitidos de un emisor a un receptor. Aún cuando en la naturaleza se pueden identificar ciertos lenguajes, los seres humanos hemos desarrollado lenguajes de diversos tipos y gran complejidad.

Los lenguajes constan principalmente de la gramática, la cual trata sobre la construcción del lenguaje, y la semántica, la cual trata sobre el significado del lenguaje.

Para que un programa en **lenguaje Python** sea válido y funcional, debe cumplir con todos estos aspectos:

* Alfabéticamente: En Python, la secuencia de comandos debe seguir el orden alfabético y utilizar los caracteres del alfabeto correctamente para formar palabras clave, nombres de variables, funciones, etc. Los identificadores deben comenzar con una letra y pueden contener letras, dígitos y guiones bajos (underscores), pero no pueden comenzar con un dígito.
* Léxicamente: En Python, la correcta escritura léxica implica seguir las reglas del lenguaje para formar palabras clave y expresiones válidas. Se deben utilizar las palabras clave reservadas de Python, como if, else, for, while, etc., de acuerdo con las reglas léxicas.
* Sintácticamente: En Python, la estructura sintáctica correcta implica utilizar la indentación adecuada para delimitar bloques de código y seguir la estructura de las declaraciones y expresiones del lenguaje. Los dos puntos (:) y la indentación son fundamentales para definir bloques de código en Python.
* Semánticamente: En Python, la corrección semántica se refiere a que el programa debe tener sentido y funcionar de acuerdo con la lógica y los objetivos deseados. Los algoritmos y las operaciones deben estar bien diseñados para producir los resultados esperados.

Durante el siglo XX, científicos como Alan Turing y Lorenzo Church fundaron las bases del cómputo, la programación y sus lenguajes. Los lenguajes de programación actuales, a diferencia de los lenguajes humanos tienen una morfología rígida y simplificada con el fin de ejecutar instrucciones específicas en los sistemas de cómputo.

**Lenguajes compilados e interpretados.**

• Codigo abierto

• Codigo cerrado - compilario

Existen dos formas de transformar un programa de un lenguaje de programación de alto

nivel a un lenguaje de máquina:

**Compilación:** el código fuente se traduce una vez (sin embargo, este proceso debe repetirse cada vez que modifiquemos el código fuente) lenguajes de alto nivel a lenguaje ensamblador o lenguaje máquina, ejemplo, un archivo con extensión .exe en Microsoft Windows; este último será el distribuido a los usuarios finales. El programa que realiza la traducción se llama compilador.

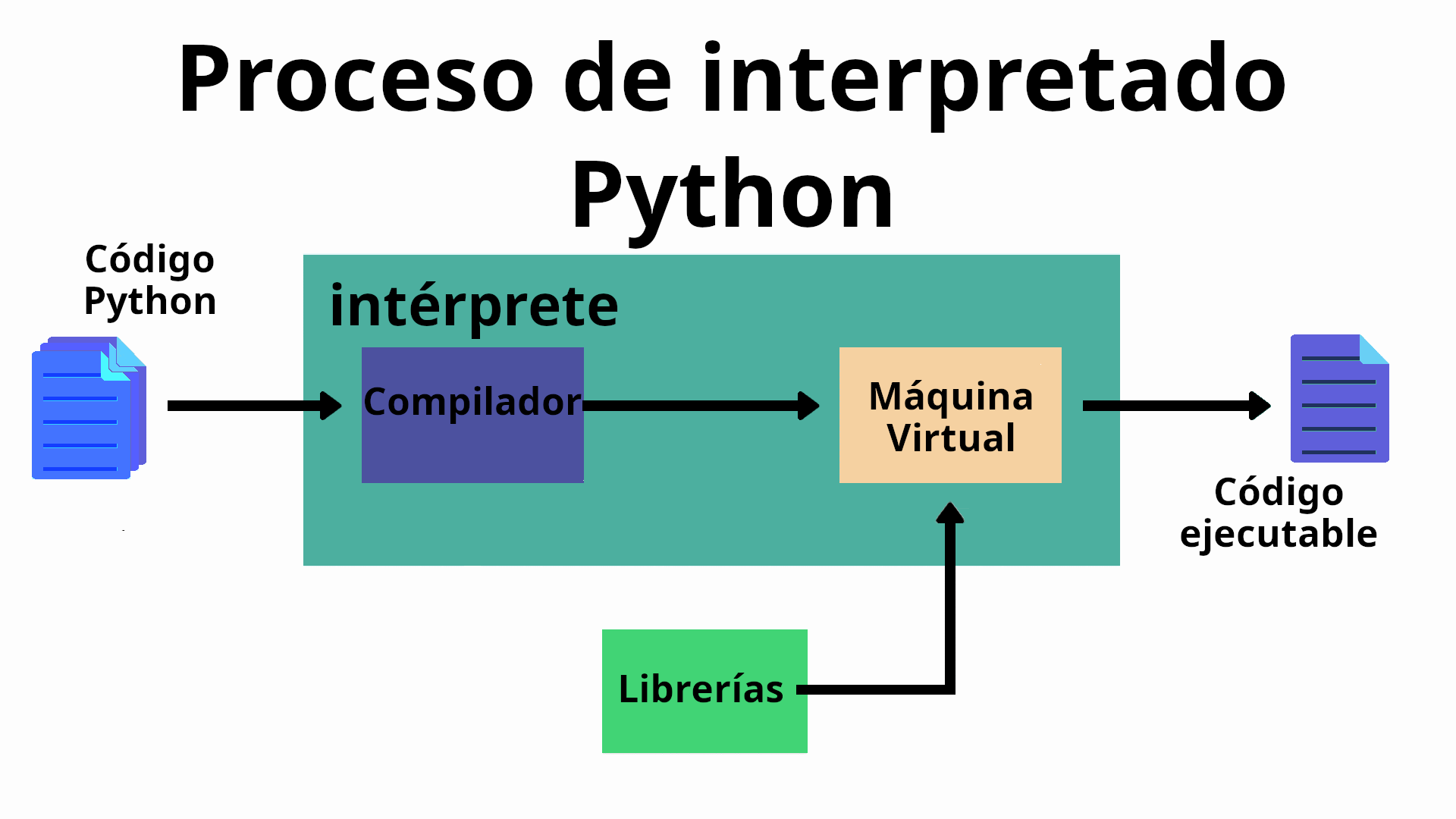
Por lo general los lenguajes compilados son más rápidos y consumen menos recursos que los lenguajes interpretados en vista de que el archivo resultante es código de bajo nivel, mientras que los lenguajes interpretados deben seguir un proceso a través de varios niveles de abstracción hasta que las instrucciones son ejecutadas por el sistema.

El script se compila en su totalidad creando un archivo de ejecución o de lectura directa por el cpu, Ej un .exe

**Interpretación:** cualquiera puede traducir el programa cada vez que se ejecute; el programa que realiza este tipo de transformación se denomina intérprete, ya que interpreta el código cada vez que este deba realizarse; esto significa que el usuario final también necesitará del programa intérprete para ejecutar nuestro programa.

En script que se interpreta lo hace a medida que el usuario lo requiere, no en la totalidad, en por ejemplo funciones donde el usuario no desea entrar el cpu nunca recibirá el código correspondiente. No se crea otro archivo

La programación es el acto de establecer una secuencia de instrucciones con la cual se causará el efecto deseado.



Cuando se ejecuta el código fuente de Python, el intérprete de Python compila el código en bytecode. El bytecode es una representación intermedia del programa que es independiente de la arquitectura de la máquina. No se puede leer directamente, pero el intérprete de Python lo entiende luego es necesario compilar el bytecode específicamente para la arquitectura de la máquina en la que se está ejecutando el programa. Esto implica que se utilice un compilador Just-in-Time (JIT) o un compilador AOT (Ahead-of-Time) que traduzca el bytecode a código de máquina adecuado para esa arquitectura. Una vez que se ha generado el código de máquina, la CPU de la máquina puede ejecutarlo directamente para producir los resultados.

El proceso de convertir el bytecode a código de máquina generalmente se realiza internamente por el intérprete de Python y los programadores normalmente no necesitan ocuparse por este proceso, ya que el intérprete se encarga de todo en tiempo de ejecución.

**Python es un lenguaje interpretado de alto nivel.**

**ver apunte “Compiladores vs Interpretes.pdf”**

### **Lenguajes de alto y bajo nivel.**

Los lenguajes de programación un un principio se pueden clasificar en dos categorías principales: lenguajes de alto nivel y lenguajes de bajo nivel. Estas categorías se refieren al nivel de abstracción que ofrecen al programador y su proximidad al lenguaje de máquina.

El avance de la tecnología redefine en nivel de abstracción y de cercanía al lenguaje humano, por lo que hoy hay una gama de grises entre assembly y Python

**Lenguajes de bajo nivel:**

Los lenguajes de bajo nivel constan de un conjunto básico de instrucciones que son ejecutados directamente por la unidad de procesamiento de un sistema de cómputo, tal como es el caso del lenguaje ensamblador. Dichos lenguajes están ligados intrínsecamente al tipo de procesador que los ejecuta y resultan ser muy complicados de elaborar e interpretar por las personas.

Los lenguajes de bajo nivel son aquellos que están más cerca del lenguaje de máquina y la arquitectura del hardware. Estos lenguajes requieren un mayor nivel de detalle y conocimiento sobre la computadora subyacente, ya que permiten un control más directo sobre el hardware y los recursos del sistema.

Ejemplos de lenguajes de bajo nivel son Assembly, C, C++, y algunos aspectos de otros lenguajes como C con sus características de punteros y acceso a memoria directa.

**Ventajas de los lenguajes de bajo nivel:**

* Permiten un control más fino sobre el hardware, lo que puede ser útil en situaciones específicas donde se necesita un alto rendimiento o acceso a características de bajo nivel.
* Son más eficientes en el uso de recursos de la computadora, ya que el código se traduce directamente a instrucciones de máquina.

**Lenguajes de alto nivel:**

Por su parte, los lenguajes de alto nivel son más accesibles para el ser humano e incluso menos dependientes del tipo de hardware, pero deben de ser a su vez traducidos a lenguaje de de bajo nivel. Se diseñan para que sean más fáciles de entender y utilizar por los programadores. Estos lenguajes se caracterizan por tener una sintaxis más cercana al lenguaje humano y ofrecer abstracciones de alto nivel que permiten expresar tareas y algoritmos de manera más natural. Esto significa que los programadores pueden concentrarse en resolver problemas sin preocuparse por los detalles específicos de la arquitectura de la computadora.

Ejemplos de lenguajes de alto nivel son Python, Java, C#, JavaScript, Ruby, y muchos otros. Estos lenguajes se utilizan ampliamente para el desarrollo de aplicaciones, sitios web, software empresarial y una variedad de otras soluciones.

**Ventajas de los lenguajes de alto nivel:**

* Facilitan la programación y la resolución de problemas.
* Son más portables, ya que están diseñados para ser independientes de la arquitectura de la computadora.
* Ofrecen bibliotecas y frameworks que facilitan el desarrollo de software.

**Lenguajes de nivel medio y Limites**

El límite entre lenguajes de alto y bajo nivel no es absoluto y puede variar según el contexto y la perspectiva. Algunos lenguajes, como C, se consideran de "nivel intermedio" porque ofrecen un equilibrio entre la abstracción y la eficiencia. Además, los avances en la tecnología de compiladores e intérpretes han permitido a algunos lenguajes de alto nivel generar código muy eficiente, acercándose al rendimiento de los lenguajes de bajo nivel.

**Resumen:**

**Lenguajes de Alto Nivel:**

|  |
| --- |
| **Abstracción:** Los lenguajes de alto nivel ofrecen un alto nivel de abstracción, lo que significa que permiten a los programadores escribir código de manera más cercana al lenguaje humano. Estos lenguajes utilizan palabras clave y estructuras que son fáciles de entender y expresivas.  **Eficiencia:** Por lo general se tiene una menor eficiencia en el uso de RAM, CPU, GPU, Puertos y demás parte del hardward.  **Portabilidad:** El código escrito en un lenguaje de alto nivel es más portátil, lo que significa que se puede ejecutar en diferentes plataformas con relativamente pocos cambios. La portabilidad se logra mediante la utilización de compiladores o intérpretes que traducen el código de alto nivel a código de máquina específico de la plataforma en tiempo de ejecución.  **Facilidad de Desarrollo:** Los lenguajes de alto nivel están diseñados para facilitar el desarrollo de software al proporcionar abstracciones que simplifican tareas complejas. Esto permite a los programadores concentrarse en la lógica del programa en lugar de preocuparse por detalles de bajo nivel. |

**Lenguajes de Bajo Nivel:**

|  |
| --- |
| **Abstracción:** Los lenguajes de bajo nivel ofrecen un nivel de abstracción más bajo, lo que significa que están más cerca del lenguaje de la máquina y son menos legibles y expresivos para los humanos. Los programadores deben lidiar con detalles de bajo nivel, como la gestión de memoria y registros de la CPU.  **Eficiencia:** Los lenguajes de bajo nivel suelen ser más eficientes en términos de uso de recursos y rendimiento, ya que permiten un control más preciso sobre el hardware subyacente.  **Portabilidad Limitada:** El código escrito en un lenguaje de bajo nivel suele ser menos portátil, ya que está más vinculado a la arquitectura y al sistema operativo específicos. Esto significa que a menudo es necesario realizar modificaciones significativas para que el código se ejecute en diferentes plataformas. |

**Herramientas de desarrollo:**

* Un editor de texto donde vamos a escribir el código (debe tener algunas características especiales, no disponibles en herramientas simples); este editor dedicado tiene más funcionalidades que el estándar del sistema operativo.
* Una consola en la que podamos ejecutar nuestro programa y frenarlo por la fuerza cuando se salga de control.
* Una herramienta llamada debugger, capaz de ejecutar tu código paso a paso y te permite inspeccionarlo en cada momento de su ejecución.
* Por suerte existen herramientas que aglomeran todo en una sola aplicación, estos son los IDLE (del inglés Integrated Development and Learning Environment) o Entornos Integrados de Desarrollo y Aprendizaje.

**Implementaciones:**

Python no sólo ha sido portado a diversos sistemas operativos, sino que existen implementaciones específicas del lenguaje. Algunas implementaciones más populares de Python son:

* CPython o es la implementación común de Python.
* [PyPy](http://pypy.org/) es una implementación de Python enfocada en optimizar su velocidad de ejecución y eficiencia en el uso de la memoria entre otras cosas.
* [Jython](http://www.jython.org/) es una implementación de Python sobre la máquina virtual de Java (JVM).
* [IronPython](http://ironpython.net/) es una implementación de Python para .NET.

● Ventajas y desventajas

● Ecosistema Python y Comunidad – Librerías extendidas

● Descarga – Opensource

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

<https://www.anaconda.com/download>

● Instalación, configuración y hardware necesario

python desde consola

anaconda

● Actualizaciones

<https://pypi.org/>

**Preparados, listos, a instalar!!!!!!!!!!**

Python cuenta con un gran número de bibliotecas (cerca de medio millon para mediados del 2023), cuyo repositorio oficial es PyPI (Python Package Index) <https://pypi.org/>, que facilitan la programación de aplicaciones complejas.

**cmd »** python -m pip install --upgrade pip wheel setuptools

**anaconda prompt»** conda install python=3.1y.xx#< ------ ver disponibilidad en python.org

conda update conda

conda update -n base -c defaults conda

● Instalaciones

**cmd »** python -m pip install colorama

beautifulsoup4

html5lib

requests

Scrapy

Pandas

Numpy

matplotlib

**anaconda prompt»** conda install requests

conda list

**cmd »** python -m pip install jupyterthemes

**anaconda prompt»** conda install -c conda-forge jupyterthemes

En Jupyter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | !jt -l |   .   |  | | --- | | Available Themes:  chesterish  grade3  gruvboxd  gruvboxl  monokai  oceans16  onedork  solarizedd  solarizedl |   .   |  | | --- | | !jt -t <theme\_name># Elijamos el tema de onedork. !jt -t onedork # o chesterish  #------------------------------------------  !jt -t <theme\_name> -T -N -kl# Con el tema de onedork, a estos les gustaría eso.  !jt -t onedork -T -N -kl# o chesterish |   .   |  | | --- | | !jt -t chesterish -T -N -kl |   . |

Reinicia el motor jupyter y….…...voila!!!!!

**cmd prompt»**

|  |
| --- |
| python -m pip install ipykernel  python -m ipykernel install --user |

**anaconda prompt»**

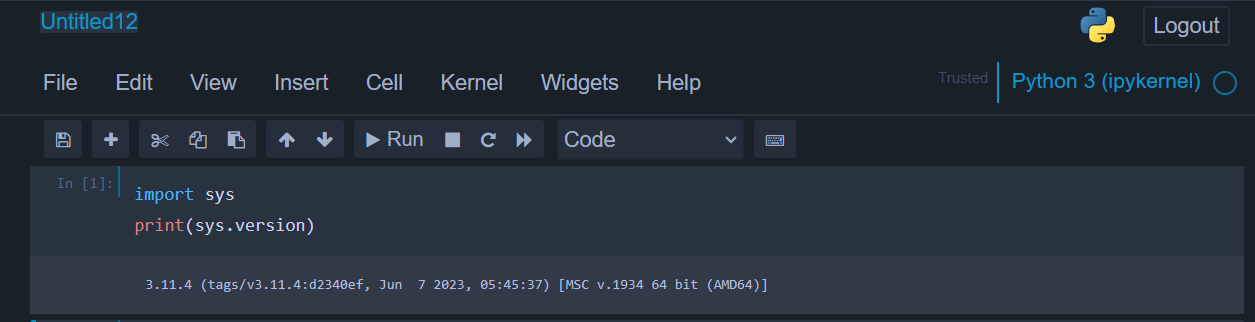
|  |
| --- |
| conda install -c conda-forge jupyterthems  conda update jupyter  conda update python --all  conda update --all  conda update -n base -c defaults conda  conda config --add channels conda-forge  conda config --set channel\_priority strict  conda create -n pypy pypy  conda activate pypy |

Reinicia Jupyter por segunda vez

|  |
| --- |
| #Obtener un listado de los módulos incorporados disponibles.  import sys  nombres\_modulos = sorted(sys.builtin\_module\_names)  nombres\_modulos = ', '.join(nombres\_modulos)  print(nombres\_modulos) |

**En Jupyter**

Original de screen capture

Formato para poder copiar y pegar en tu Jupyter o visual studio code

|  |  |  |  |
| --- | --- | --- | --- |
| ·Código Python   |  | | --- | | import sys  print(sys.version)  print(sys.version\_info)  #print(sys.copyright) |   Salida esperada por consola   |  | | --- | | 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]  sys.version\_info(major=3, minor=12, micro=0, releaselevel='final', serial=0) |   ·Código Python   |  | | --- | | import platform  import os  print(os.name)  print(platform.system())  print(platform.release())  import multiprocessing  print(f"{multiprocessing.cpu\_count()=}") |   . |

·

Todas las computadoras responden a una serie de comandos predeterminados.

Este conjunto de comandos se conoce como "Instruction List", abreviado IL.

## Entornos interactivos, guiones y ****'Hola Mundo'****.

Debido a que Python es un lenguaje interpretado, es posible utilizarlo mediante un entorno interactivo (shell) o mediante el uso de guiones (scripts).

### El entorno interactivo.

El entorno interactivo (shell) de Python se ejecuta desde una terminal de texto.

|  |
| --- |
| cursos@openSuse:-$ python3  Python 3.11.4 (default, Jul 02 2024, 09:44:23) [GCC] on linux  Type "help", "copyright", "credits" or "license" for more information.  >>> |

·En el caso de los sistemas basados en UNIX, como GNU/Linux \\*BSD y Mac OS X, es necesario abrir una terminal de texto e invocar el shell de la siguiente manera:

### Ejecución de guiones (scripts).

Los scripts de Python son archivos de texto que contienen código que será leído y ejecutado por el intérprete de Python. Dichos archivos tiene la extensión .py . Algunos editores de texto cuentan con reconocimiento de la sintaxis de Python y utilizan colores para identificar sus componentes.

#### El script holamundo.py

|  |
| --- |
| #! /usr/bin/python3  print("Hola, Mundo.") |

La ejecución del script en entornos basados en UNIX requiere que se indique la ruta en la que se encuentra el intérprete de Python mediante el texto escrito la primera línea de holamundo.py. En la gran mayoría de las plataformas basadas en UNIX el intérprete de Python 3 se encuentra en /usr/bin/python3. Esta línea no afecta la ejecución del script en Windows, por lo que se recomienda incluirla siempre.

#### Ejecución en entornos basados en UNIX.

Si se dessea ejecutar cualquier script en entornos basados en UNIX, es necesario que el script cuente con los permisos necesarios. Para asignar permisos de ejecución a un archivo en \\*UNIX se utiliza el comando:

|  |
| --- |
| pc@linux:-$ chmod +x <ruta del archivo> |

Por ejemplo, el siguiente comando en la terminal asignará permisos de ejecución al script holamundo.py

|  |
| --- |
| pc@linux:-$ chmod +x holamundo.py |

Y el script se ejecutará de la siguiente forma, suponiendo que se encuentra en el mismo directorio de trabajo de la terminal.

|  |
| --- |
| pc@linux:-$ ./holamundo.py  Hola Mundo  pc@linux:-$ |

Además de la forma previa, es posible ejecutar el script mendiante el intérprete de Python 3, sin necesidad de otorgarle al archivo permisos de ejecución.

|  |
| --- |
| pc@linux:-$ python3 holamundo.py  Hola Mundo  pc@linux:-$ |

### **Ejecución en Windows.**

En el caso de Windows, el sistema relacionará a los archivos con la extensión .py con el intérprete de Python, por lo que con hacer doble click en el archivo, éste se ejecutará y tan pronto termine, cerrará la terminal.

Cuando se ejecute el script holamundo.py en Windows, se abrirá y cerrará una terminal de forma casi inmediata.

|  |
| --- |
| Microsoft Windows [Versión 10.0.19045.2194]  (c) Microsoft Corporation. Todos los derechos reservados.  C:\Users\cursos>python  Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32  Type "help", "copyright", "credits" or "license" for more information.  >>> |

### **Manejo de IDE Jupyter:**

### **Ejecución en Jupyter.**

Una Jupyter Notebook es una aplicación web que permite crear y compartir documentos que contienen código, visualizaciones y texto explicativo. El nombre "Jupyter" es un acrónimo que se refiere a los lenguajes de programación que soporta: Julia, Python y R.

En un Jupyter Notebook, el usuario puede escribir código en una variedad de lenguajes de programación, incluyendo Python, R, Julia, y otros, y ejecutarlo en tiempo real. Además, los resultados de la ejecución del código pueden ser visualizados en gráficos, tablas, y otros formatos.

Los Jupyter Notebooks son populares entre los científicos de datos, los ingenieros de software, y otros profesionales que necesitan documentar y compartir sus resultados de investigación y análisis de datos. Los notebooks son fáciles de compartir en línea y permiten a los usuarios trabajar juntos en proyectos de manera colaborativa, una implementación muy popular el [google colaboratory](https://colab.research.google.com/) .

Jupyter puede ejecutar scripts mediante el "comando mágico" %run.

# **Jupyter** es un IDE que trabaja en un formato especial. Notebooks, enfocado a manejo de datos, principalmente en los lenguajes **Ju**lia, **Py**thon y **R**.

# La diferencia mas destacada con un IDE de programación es su ejecución por burbujas o celdas.

Es un IDE que no facilita el trabajo en Funciones ni en POO. Es parte de un paquete cuyas librerías incorporadas son las preferidas para datacience, IA, BI, etc.

**El manejo de celdas:**

Jupyter Notebook ofrece varios tipos de celdas que te permiten trabajar con diferentes tipos de contenido. Aquí están los tipos de celdas más comunes y sus usos:

**Celdas de Código:**

Estas celdas están diseñadas para escribir y ejecutar código. Puedes escribir código en lenguajes como Python, R, Julia y otros. Cuando ejecutas una celda de código, el resultado se muestra debajo de la celda.

**Celdas de Markdown:**

Las celdas de Markdown te permiten agregar texto formateado utilizando el lenguaje Markdown. Puedes crear documentación, comentarios, explicaciones y otros tipos de texto enriquecido utilizando Markdown.

En breve haremos una descripción de este tipo de celda

**Celdas de Títulos y Encabezados:**

Estas celdas se utilizan para agregar títulos y encabezados a tus cuadernos. Puedes usar encabezados para dividir tu contenido en secciones y darle estructura a tu cuaderno.

**Celdas de Raw:**

Las celdas de tipo "Raw" son celdas de texto que no se procesan ni se interpretan de ninguna manera. Esto es útil si necesitas incluir contenido que no debe ser formateado ni ejecutado, como código que no debe ser interpretado o instrucciones específicas para otros usuarios.

**Celdas de Salida:**

Las celdas de salida muestran los resultados de la ejecución de celdas de código. Pueden contener texto, gráficos, tablas u otros tipos de resultados generados por el código.

**Celdas de Código y Salida:**

Estas celdas combinan una celda de código con su salida en una sola celda. Esto puede ser útil para mostrar el código y los resultados juntos, lo que facilita la comprensión del proceso.

**Celdas de Código Oculto:**

Estas celdas contienen código que se ejecuta en segundo plano, pero su contenido y resultados no se muestran en el cuaderno. Son útiles para definir funciones, variables u operaciones que son necesarias para el flujo de trabajo, pero que no deben ser visibles en el cuaderno final.

**Celdas de Widgets Interactivos:**

Jupyter Notebook admite widgets interactivos que permiten a los usuarios interactuar con el código y los resultados. Puedes crear controles deslizantes, botones y otros elementos interactivos para ajustar parámetros y ver cómo cambian los resultados.

**Markdown y HTML**

**Jupyter soporta markdown y etiquetas html. Lo veremos en ese orden**

Los lenguajes de marcado no son lenguajes de programación. Los primeros solo sirven para determinar de qué forma debe estructurarse un texto. Los lenguajes de programación, por el contrario, se caracterizan por presentar bucles y variables y, por lo tanto, son la base para crear software.

Markdown es especialmente interesante para los usuarios que no tienen experiencia en TIC ni conocimientos de diseño web, pero que suelen escribir textos en Internet con regularidad, como, por ejemplo, los bloggers que trabajan con sistemas de gestión de contenidos (CMS). Markdown no pretende ser un sustituto de HTML, ya que su rango de opciones de formateo es demasiado reducido.

Para generar un nuevo párrafo en Markdown simplemente separa el texto mediante una línea en blanco (pulsando dos veces enter).

Al igual que sucede con HTML, Markdown no soporta dobles líneas en blanco, así que si intentas generarlas estas se convertirán en una sola al procesarse.

Para realizar un salto de línea y empezar una frase en una línea siguiente dentro del mismo párrafo, tendrás que pulsar dos veces la barra espaciadora antes de pulsar una vez enter.

Crear una Celda de Markdown:

En un cuaderno de Jupyter, puedes crear una nueva celda haciendo clic en el botón "+" en la barra de herramientas o usando el atajo de teclado "B" para crear una celda debajo de la celda seleccionada. Luego, cambia el tipo de celda a "Markdown" en el menú desplegable de tipo de celda en la barra de herramientas o con el atajo de teclado "M" (depende del navegador el poder usar o no los atajos).

Ingresar Contenido en Markdown:

Una vez que hayas creado una celda de Markdown, puedes ingresar tu contenido en Markdown directamente en la celda. Se usan líneas vacías para separar los párrafos, si contiene caracteres en blanco, como tabulaciones o espacios, Jupyter los ignorará e interpretará que la línea está vacía. Si deseas insertar un salto de línea como el de la etiqueta, introduce dos espacios al final de la línea.

**Markdown**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| · Encabezado markdown  Los numerales # son uno de los métodos utilizados en Markdown para crear encabezados. Debes usarlos añadiendo uno por cada nivel.   |  | | --- | | # (Header 1, title)  ## (Header 2, major headings)  ### (Header 3, subheadings)  #### (Header 4)  ##### (Header 5) |   · Encabezado +-   |  | | --- | | Header 1  =  Header 2  - |   ·hipervínculos o Links   |  | | --- | | [hipervínculos del enlace](http(s)://URL\_del\_hipervínculos/)  [click para Enlace](/ruta\_o\_path/archivo) |   ·Imagenes   |  | | --- | | ![Texto alternativo](http(s)://URL\_de\_la\_imagen)  ![Istea]()  ![Texto alternativo](/ruta\_o\_path/imagen) |   ·   |  | | --- | | ![nombre de la imagen1][img1]  ![nombre de la imagen2][img2]  [img1]: /ruta\_o\_path/imagen1.jpg "Título imagen 1"  [img2]: /ruta\_o\_path/imagen2.jpg "Título imagen 2" |   · |

·

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ·Strings Texto / párrafos  Simplemente tendrás que envolver palabras o textos en éstos símbolos para conseguir cursivas o negritas.   |  | | --- | | \* Esto es un elemento de lista con viñeta  \* Otro elemento en la lista |   ·   |  | | --- | | Esta \*palabra\* va en cursiva.  Esta \_palabra\_ va en cursiva.  Esta \*\*palabra\*\* va en negrita.  Esta \_\_palabra\_\_ va en negrita.  Esta \*\*\*palabra\*\*\* va en cursiva y negrita.  Esta \_\_\_palabra\_\_\_ va en cursiva y negrita.  Este texto ~~está tachado~~ . |   ·   |  | | --- | | >Este es un \*\*fragmento con negrilla\*\*.  >El fragmento continúa aquí.  >Este es otro \*\*fragmento con negrilla  bastante  mas largo  Para marcar un fragmento de texto como una cita en Markdown, puedes crear las llamadas citas en bloque, utilizando el signo de mayor que (>). Tienes dos opciones: marcar cada línea individual con este carácter o insertar únicamente uno al principio del párrafo de la cita sangrada y marcar el fin de la cita introduciendo una línea en blanco. La cita en bloque, a su vez, puede ser formateada con otros elementos.\*\*.  Este fragmento continúa en la siguiente línea sin negrillas.  Esta línea ya no está sangrada. |   ·Citas  Las citas se generar utilizando el carácter mayor que `>` al comienzo del bloque de texto.   |  | | --- | | > Un país, una civilización se puede juzgar por la forma en que trata a sus animales. — Mahatma Gandhi |   ·  Si la cita en cuestión se compone de varios párrafos, deberás añadir el mismo símbolo > al comienzo de cada uno de ellos.   |  | | --- | | > Creo que los animales ven en el hombre un ser igual a ellos que ha perdido de forma extraordinariamente peligrosa el sano intelecto animal.  > Es decir, que ven en él al animal irracional, al animal que ríe, al animal que llora, al animal infeliz. — Friedrich Nietzsche |   Recuerda separar los saltos de línea con `>`, o `> >` si te encuentras dentro de la cita anidada; para crear párrafos dentro del mismo bloque de cita.  ·Notas al pie  Se escribe un número de nota en el texto y refiérelo a una nota al pie al final de la página: se generará automáticamente la línea correspondiente. El número de nota también se formatea como un hipervínculo, que nos lleva directamente a la nota al pie correspondiente al hacer clic en él. Para utilizar esta función automática, deberás insertar el número de la nota detrás de la palabra que desees. Para ello, escribe entre corchetes un acento circunflejo y, después, el número.  El número que utilices (se admiten también otros elementos) es irrelevante. Al igual que con la creación de listas, Markdown realiza el conteo automáticamente. Si es importante que vuelvas a introducir la misma denominación. Para ello, pon el mismo número en una nueva línea, de nuevo entre corchetes y precedido por un acento circunflejo, inserta dos puntos y, luego, escribe el texto de la nota. Este, a su vez, también podrá formatearse de todas las maneras posibles y ocupar varias líneas.   |  | | --- | | Puedes colocar [^1] notas en el pie de página [^2] fácilmente.  [^1]: Aquí encuentras el texto de la nota al pie de página.  [^2]: \*\*Las notas de pie de página\*\* pueden \*formatearse\* también. |   Estas pueden ocupar varias líneas. |

·

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Lista no ordenada utiliza el signo de más(+) o un guion(-) o un asterisco(\*),obtendrás el mismo resultado.  Para generar listas anidadas dentro de otras, simplemente tendrás que añadir cuatro espacios en blanco antes del siguiente \*, - o +.   |  | | --- | | - Elemento de la lista 1  + Elemento de la lista 2  \* Elemento de la lista 3 |   ·Lista ordenada deberás introducir un número con un punto.   |  | | --- | | 1. Elemento de la lista 1  2. Elemento de la lista 2  3. Elemento de la lista 3 |   ·Si lo que quieres es crear una lista ordenada, deberás introducir un número con un punto directamente después. No podrás modificar el orden   |  | | --- | | 1. Elemento de la lista 1  2. Elemento de la lista 2  2. Elemento de la lista 2  2. Elemento de la lista 2  55. Elemento de la lista 55 |   Para crear listas ordenadas debes utilizar la sintaxis de tipo: ```número.```, por ejemplo, ```1.```. Al igual que ocurre con las listas desordenadas, también podrás anidarlas o combinarlas. Verificar los espacios  ·   |  | | --- | | 1. Elemento de lista 1  2. Elemento de lista 2  - Elemento de lista 3  - Elemento de lista 4  A. Elemento de lista 5  B. Elemento de lista 6 |   ·   |  | | --- | | - [1 - Tema 1](#1)  - [2 - Tema 2](#2)  - [3 - Tema 3](#3)  - [3.1 - Subtema del Tema 3](#3-1)  - [3.2 - Subtema del Tema 3](#3-2)  - [4 - Tema 4](#4) |   · |

·

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ·Fórmulas matemáticas  Existiendo la posibilidad de incluir dentro de estas fórmulas matemáticas escritas con sintaxis de LaTeX. Para agregar una ecuación se debe escribir dentro de un bloque con `$`, incorporando de esta manera las ecuaciones en el mismo párrafo, o `$$`, con lo que la ecuación se presentará en una nueva línea y centrada.  Las sintaxis para escribir ecuaciones con LaTeX ofrecen la posibilidad de usar entornos para desarrollar fórmulas complejas. Entornos que comienza con `\begin{} y finaliza con \end{}`, escribiendo dentro de los corchetes el nombre de entorno. Existen dos con los que se pueden usar para alienar ecuaciones multilínea: \*\*align\*\* y \*\*aligned\*\*.  En ambos casos dentro del entorno se pueden agregar todas las ecuaciones que se desee y para separarlas mediante un salto de línea se debe escribir `\\`. Además, mediante el símbolo `&` es posible alinear las ecuaciones, lo que marca una posición fija para todas las líneas.   |  | | --- | | $$  \begin{align}  x + y &= 10\\  x - y &= 2\\  2x +y- y &= 12\\  x &=12/2 \\  x&=6\\  x + y &= 10\\  y &= 10-6\\  x&=4\\  \end{align}  $$ |   ·Lo que produce una fórmula como la siguiente en la que los símbolos de igualdad se encuentran alineados.   |  | | --- | | 𝑥+𝑦=10  𝑥−𝑦=2  2𝑥+𝑦−𝑦=12  𝑥=12 /2  𝑥=6  𝑥+𝑦=10  y=10-6  y=4 |   ·   |  | | --- | | $$  \begin{align}  x + y &= 10 – x\\  y &= 10 – x = 4\\  y &= 4\\  \end{align}  $$ | | x + y = 10  y = 10 – x =4  y = 4 |   ·   |  |  | | --- | --- | | $$  \begin{align}  x + y &= 10 & a + b &= 5\\  x - y &= 2 & a - b &= 1\\  2\*x &= 10+2 & 2\*a &= 5+1 \\  x &= 12/2 & a &= 6/2\\  x &= 6 & a &= 3\\  y &= 10-6 & b &= 5-3 \\  y &= 4 & b &= 2 \\  \end{align}  $$ | | | x + y = 10  x - y = 2  2\*x = 10+2  2\*x = 12/2  x = 6  x + y = 10  y = 10 -6  y = 4 | a + b = 5  a - b = 1  2\*a = 5+1  2\*a = 6/2  a = 3  a + b = 5  b = 5 -3  b = 2 |   ·En esta ecuación es importante notar cómo se alinean las expresiones. Algo que es importante a la hora de trabajar con sistemas complejos. Lo que se encuentra antes del primer `&` se aliena a la derecha y después a la izquierda. Por otro lado, lo que se encuentra después del segundo se vuelve a alinear a la derecha del tercer símbolo.  Otra opción que existe en los entornos align y aligned es la posibilidad de incluir aclaraciones mediante comandos como `\text{}`. Así se puede explicar la resolución del sistema de ecuaciones tal como se muestra en el siguiente código.  Podemos incluir múltiples ecuaciones en una línea, para lo que se puede usar más de una vez el símbolo &. Por ejemplo, se pueden poner dos sistemas de ecuaciones uno al lado del otro mediante el siguiente código.   |  |  | | --- | --- | | $$  \begin{align}  y &= 10 - x && \text{Se reemplaza x por su valor} \\  y &= 10 - 6 = 4 && \text{El resultado es y = 4}\\  \end{align}  $$ | | | y = 10 - x  y = 10 - 6 = 4 | Se reemplaza x por su valor  El resultado es y = 4 |   ·  Nótese que en este caso se ha empleado && en lugar de & lo que se hace para que los textos se alineen a la izquierda y no a la derecha.  Algunas otras cosas útiles a la hora de hacer ecuaciones es usar letras griegas, para ello, lo hacemos de la siguiente forma: `$\alpha$` lo que se ve como: $\alpha$  También podríamos necesitar elevar una variable o un número a un exponente, como por ejemplo: $x^{2}$, para ello, usamos este tipo de expresión: `$x^{2}$` donde los `{}` encierran todo el argumento que se elevará.  En cambio, si necesitamos hacer subíndices, pordemos usar el guión bajo `\_`. Por ejemplo: $x\_1$, esto se escribe como `$x\_1$`. |

·

|  |  |
| --- | --- |
| Tablas   |  | | --- | | | Company | Contact | Country |  | --- | --- | --- |  | Alfreds Futterkiste | Maria Anders | Germany |  | Centro comercial Moctezuma | Francisco Chang | Mexico |  | --- | --- | --- | |   · |

Aquí hay un ejemplo de una tabla básica en Markdown:

| Column 1 | Column 2 | Column 3 |

|:--------:|:--------:|:--------:|

| Row 1 | Row 1 | Row 1 |

| Row 2 | Row 2 | Row 2 |

| Row 3 | Row 3 | Row 3 |

Este código crea una tabla con tres columnas y tres filas. los :--------: la sintaxis en la segunda fila indica que cada columna debe estar centrada.

## **Opciones de formato**

Las tablas de marcado admiten varias opciones de formato que le permiten personalizar la apariencia de su tabla. Estas son algunas de las opciones de formato más comunes:

### **Alineación de columna**

Como vimos en el ejemplo básico de sintaxis, puede usar dos puntos y guiones para definir la alineación para cada columna. Estas son las opciones disponibles:

* :--: Alinear a la izquierda la columna
* :--:: Centrar la columna
* --:: Alinear a la derecha la columna

### **Alineación celular**

También puede alinear el contenido de celdas individuales dentro de una columna usando la misma sintaxis que la anterior. Por ejemplo, para centrar el contenido de una celda, usaría | :--: |.

### **Revestimiento de celda**

De forma predeterminada, las tablas Markdown no tienen relleno entre el contenido de la celda y el borde. Para agregar relleno, puede agregar espacios entre las barras verticales (|). Por ejemplo, | Cell Contents | no tendría relleno, mientras | Cell Contents | tendría relleno.

### **Fronteras**

Por defecto, las tablas de Markdown no tienen bordes. Para agregar bordes, puede usar una combinación de guiones y barras verticales.

|  |  |  |
| --- | --- | --- |
| Column 1 | Column 2 | Column 3 |
| Row 1 | Row 1 | Row 1 |
| Row 2 | Row 2 | Row 2 |
| Row 3 | Row 3 | Row 3 |

Este código crea una tabla con bordes entre cada celda.

**Etiquetas HTML**

Solo presentaremos una mínima parte del enorme repertorio de html. No dudes en informarte usar mas elementos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| · Encabezado Html   |  | | --- | | <h1>Header 1,title<h1>  <h2>Header 2,major headings<h2>  <h3>Header 3,subheadings<h3>  <h4>Header 4<h4>  <h5>Header 5<h5> |   ·hipervínculos o Links   |  | | --- | | < a > href ="http(s)://URL\_del\_hipervínculos/" > hipervínculos del enlace < / a >  < a > href ="ruta\_o\_path/archivo" > hipervínculos del enlace < / a > |   ·Imagenes   |  | | --- | | ![Texto alternativo]()  < img > < src ="http(s)://URL\_de\_la\_imagen" ancho =" 500 "altura = 500 >  < img > < src ="<https://static.educacionit.com/common/assets/logotype-istea-fill-blue.svg>" width ="500" height=500 >  ![Texto alternativo](/ruta\_o\_path/imagen) |   ·   |  | | --- | | ![nombre de la imagen1][img1]  ![nombre de la imagen2][img2]  [img1]: /ruta\_o\_path/imagen1.jpg "Título imagen 1"  [img2]: /ruta\_o\_path/imagen2.jpg "Título imagen 2" |   · |

·

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ·Etiqueta  <dl>- Esta etiqueta se utiliza para crear una lista de definiciones si lo deseamos.  <dt>- Esta etiqueta contiene el término que queremos definir.  <dd>- Esta etiqueta contiene la definición del término que le precede.   |  | | --- | | <dl><dt>Mamifero</dt><dd> Especie de animales que maman</dd></dl>  <dl><dt>Vertebrado</dt><dd>Especie de animales que poseen huesos</dd></dl>  <dl><dt>Reptil</dt><dd>Especie de animales que se arrastran por el suelo</dd></dl> |   ·Strings Texto / párrafos  Simplemente tendrás que envolver palabras o textos en éstos símbolos para conseguir cursivas o negritas.   |  | | --- | | Esta < i >palabra</ i > va en cursiva.  Esta < b >palabra</ b > va en negrita.  Esta < i >< b >palabra</ b ></ i > va en cursiva y negrita.  Este texto <strike>está tachado</strike> |   ·Centrado   |  | | --- | | < center> Un país, una civilización se puede juzgar por la forma en que trata a sus animales. — Mahatma Gandhi<center> |   ·Colores   |  | | --- | | <font color="blue"> Un país, una civilización se puede juzgar por la forma en que trata a sus animales. — Mahatma Gandhi</font> |   ·Citas  Las citas se generar utilizando la etiqueta <blockquote>   |  | | --- | | < blockquote > Un país, una civilización se puede juzgar por la forma en que trata a sus animales. — Mahatma Gandhi |   ·   |  | | --- | | < blockquote > Creo que los animales ven en el hombre un ser igual a ellos que ha perdido de forma extraordinariamente peligrosa el sano intelecto animal.  Es decir, que ven en él al animal irracional, al animal que ríe, al animal que llora, al animal infeliz. — Friedrich Nietzsche < / blockquote > |   ·Notas al pie   |  | | --- | | Puedes colocar <footer></footer> + codigo de lick |   Estas pueden ocupar varias líneas. |

·

|  |  |  |
| --- | --- | --- |
| Lista no ordenada   |  | | --- | | < ul >  < li > Elemento de la lista 1< / li >  < li > Elemento de la lista 2< / li >  < li > Elemento de la lista 3< / li >  < / ul > |   ·Lista ordenada   |  | | --- | | < ol >  < li > Elemento de la lista 1< / li >  < li > Elemento de la lista 2< / li >  < li > Elemento de la lista 3< / li >  < / ol > |   · |

·

También podemos esconder texto opcional para que sea revisado por el usuario cuando lo desee. Por ejemplo, cuando escondemos pistas para resolver un problema:

|  |  |
| --- | --- |
| Tablas   |  | | --- | | <table>  <tr>  <th>Company</th>  <th>Contact</th>  <th>Country</th>  </tr>  <tr>  <td>Alfreds Futterkiste</td>  <td>Maria Anders</td>  <td>Germany</td>  </tr>  <tr>  <td>Centro comercial Moctezuma</td>  <td>Francisco Chang</td>  <td>Mexico</td>  </tr>  </table> |   · |

Finalmente, una funcionalidad muy útil es la de armar índices o Contenido de una Notebook. Como muestra el siguiente ejemplo, donde podemos hacer clic en el índice y nos llevará directo a ese título.

|  |  |
| --- | --- |
| Un mix entre markdown y html   |  | | --- | | ## Tabla de contenidos  <a name='3-1'></a>  ### 3.1 - Subtema del Tema 3  Contenido del Tema 3.1  <a name='3-2'></a>  ### 3.2 - Subtema del Tema 3  Contenido del Tema 3.2  <a name='4'></a>  ## 4 - Tema 4  Contenido del Tema 4 |   · |

·

|  |  |  |
| --- | --- | --- |
| ·   |  |  | | --- | --- | | A verificar por el alumno  <ol>  <li> lunes</li>  <li> martes </li>  <li> miércoles </li>  <li> jueves </li>  <li> viernes </ li>  <li> sábado </li>  <li> domingo</li>  </ol> | <ul>  <li>América</li>  <ol>  <li>América central</li>  <li>América sur</li>  <li>América norte</li>  </ol>  <li>África</li>  <ol>  <li>África Septentrional</li>  <li>África Occidental</li>  <li>África Central</li>  <li>África Oriental</li>  <li>África Meridional</li>  </ol>  <li>Antártida</li>  <ul>  <li>Antártida oriental</li>  <li>Antártida occidental</li>  </ul>  <li>Asia</li>  <li>Europa</li>  <li>Oceania</li>  </ul> |   · |

·

**Micro historia de** [**Python**](http://python.org/)

Python es un lenguaje de programación de alto nivel, interactivo e interpretado, creado por Guido Van Rossum en 1991. El nombre del lenguaje es en honor al programa cómico de la televisión británica llamado “Monty Python Flying Circus”

Python es un lenguaje de programación que Van Rossum empezó a desarrollar mientras trabajaba en CWI. A partir de la publicación de la versión 1.2 en 1995 hizo aún más accesible el código y para el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo de BeOpen Python Labs.

Hasta 2018, el desarrollo de Python estaba dirigido personalmente por Guido van Rossum y Python Software Foundation. En julio de 2018 Guido van Rossum anunció que dejaría de dirigir el desarrollo de Python. Desde 2019 el desarrollo de Python está dirigido por un consejo de dirección de cinco miembros elegidos entre los desarrolladores de Python y que se renueva anualmente.

I**mportancia del versionado. (retro compatibilidad)**

• La versión 1.0 se publicó en enero de 1994,

• La versión 1.2 se publicó en enero de 1995,

Fue este centro de investigación quien liberó, en 1995 la versión 1.2 de Python.

• La versión 2.0 se publicó en octubre de 2000.

• La versión 3.0 se publicó en diciembre de 2008.

3.10 se publicado en 2021-10-04

3.11 se publicado en 2022-10-24

3.12 se publicara en 2023-10-02 (esperada)

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

**X** corresponde a las grandes versiones de Python (1, 2 y 3), **incompatibles entre sí:**

**Y** corresponde a versiones importantes en las que se introducen novedades en el lenguaje pero manteniendo generalmente la compatibilidad pero con excepciones.

**Z** corresponde a versiones menores que se publican durante el período de mantenimiento, en las que sólo se corrigen errores durante el primer año y fallos de seguridad en los cuatro restantes.

En julio de 2018, Guido van Rossum anunció su retirada como BDOF.(dictador benévolo)

El primer consejo directivo se eligió en enero de 2019 aun con Guido. El segundo consejo directivo se eligió en diciembre de 2019 ya sin Guido

En noviembre de 2020, Guido van Rossum anunció que se incorporaba a la **Developer Division de Microsoft** para trabajar en temas relacionados con Python.

En mayo de 2021 anunció que centraría su trabajo en la mejora del rendimiento de Python.

**Retro-compatibilidad negativa**

****NOTA:**** En este documento se utilizará código escrito en Python 3, pero se discutirá sobre las diferencias entre cada versión cuando se considere pertinente.

Python 3.0 rompe la compatibilidad hacia atrás del lenguaje, ya que el código de Python 2.x no necesariamente debe correr en Python 3.0 sin modificación alguna.

Python 2.7 es una versión "de transición" que permite usar algunos elementos sintácticos tanto de Python 2 como de Python 3.

Python 3 presenta mejoras notables con respecto a Python 2, sin embargo aún existe mucho código legado compatible con Python 2.

EL soporte de Python 2 ceso a partir del año 2020.

Si bien, a la fecha, sigue siendo de código abierto, multiplataforma y se adecúa a diversos paradigmas de programación (programación funcional, programación orientada a objetos, programación procesal, etcétera). Se ha vuelto muy popular ya sea como un primer lenguaje de programación o como lenguaje seleccionado por defecto por diversos proyectos de primer nivel tales como Openstack, Blender, Google App Engine, Django, Jupyter , entre otros.

Busca el significado del acrónimo Jupyter en la tierra

Hoy es la base de la programación de Inteligencia Artificial, Data Science, Machine Learning, Deep L. Etc. Es interesante preguntarle a bart (<https://bard.google.com/u/1/>) o chatgpt (<https://chat.openai.com/>) en que lenguaje estan escritos o programados.

Por otro lado se pueden observar los repositorios de github (microsoft) y stackoverflow que lenguajes tienen mas generación de código en el mundo.

https://insights.stackoverflow.com/trends?tags=java%2Cc%2Cc%2B%2B%2Cpython%2Cc%23%2Cvb.net%2Cjavascript%2Cassembly%2Cphp%2Cperl%2Cruby%2Cswift%2Cr%2Cobjective-c

|  |
| --- |
| import platform as pl  perfil\_so = [  'architecture',  'linux\_distribution',  'mac\_ver',  'machine',  'node',  'platform',  'processor',  'python\_build',  'python\_compiler',  'python\_version',  'release',  'system',  'uname',  'version',  ]  for perfil in perfil\_so:  if hasattr(pl, perfil):  print('%s: %s' % (perfil, getattr(pl, perfil)())) |