

«Talento Tech»

Iniciación a la Programación con Python

CLASE 1



Clase N° 1 | Conceptos básicos

Temario:

- Introducción al curso: duración, objetivos, metodología, evaluación.
- Conceptos básicos: hardware, software, periféricos, historia de la informática.
- Concepto de algoritmo. Entrada-Proceso-Salida.
- Estrategias para la resolución de problemas.
- Introducción a Python: Origen, características, campo de aplicación.



Iniciación a la programación con Python.

Iniciación a la programación con Python

En este trayecto exploraremos cómo el conocimiento de un lenguaje de programación, como Python, puede abrir numerosas oportunidades laborales. Python, con su simplicidad y versatilidad, se utiliza en áreas como desarrollo web, ciencia de datos, inteligencia artificial y automatización, preparándote para enfrentar y aprovechar estas oportunidades en tu futuro profesional.

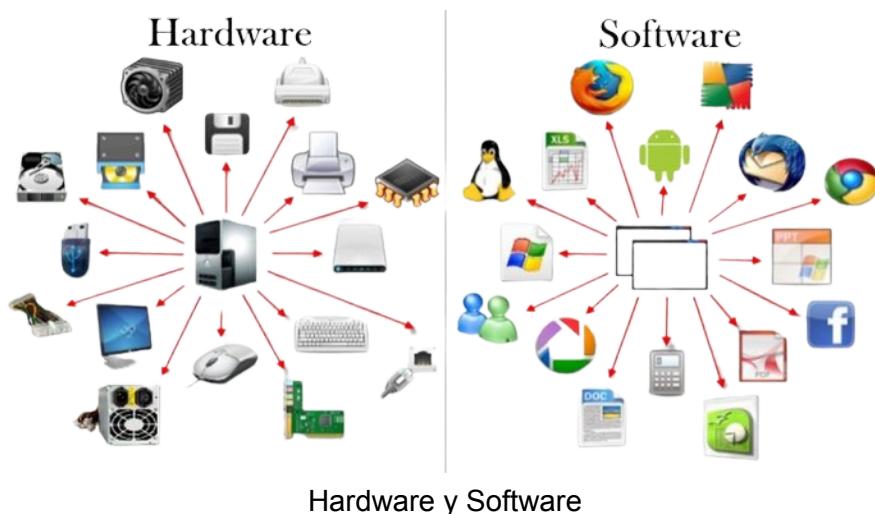
Conceptos básicos.

Conceptos básicos

En el amplio y cambiante mundo de la informática, es fundamental comprender los conceptos que forman la base de todo sistema computacional. La relación entre hardware y software es un pilar en el que se sustentan todas las tecnologías modernas, y aunque hoy en día las diferencias entre ambos son claras y bien definidas, no siempre fue así. Para los programadores principiantes y amantes de la tecnología, entender esta dicotomía es esencial, no sólo para comprender cómo interactúan estos componentes en el funcionamiento diario de los dispositivos, sino también para apreciar la evolución que ha permitido el desarrollo de las herramientas que hoy consideramos fundamentales en nuestra vida digital.



Diferencia entre hardware y software:



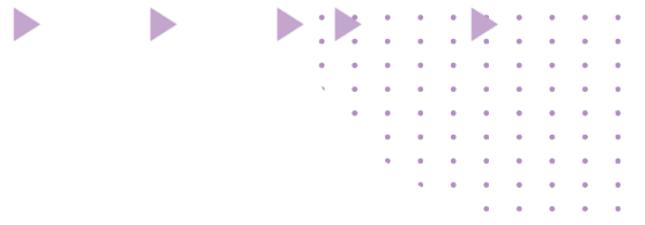
Diferencia entre hardware y software

Al principio de la era informática, la diferencia entre hardware y software no era tan evidente como en la actualidad.

El **hardware**, compuesto por los elementos físicos de un sistema informático, ha sido siempre fundamental para el funcionamiento del software. Durante los primeros años de la computación, los grandes mainframes, que ocupaban salas enteras, operaban mediante una compleja sinergia de circuitos electrónicos, cables y componentes mecánicos. Estos sistemas, aunque básicos en comparación con la tecnología actual, fueron esenciales para el desarrollo de las tecnologías futuras. Este contexto histórico es crucial para entender cómo las innovaciones en hardware han impulsado los avances en software, y viceversa.

Buenos Aires aprende

Agencia de Habilidades para el Futuro



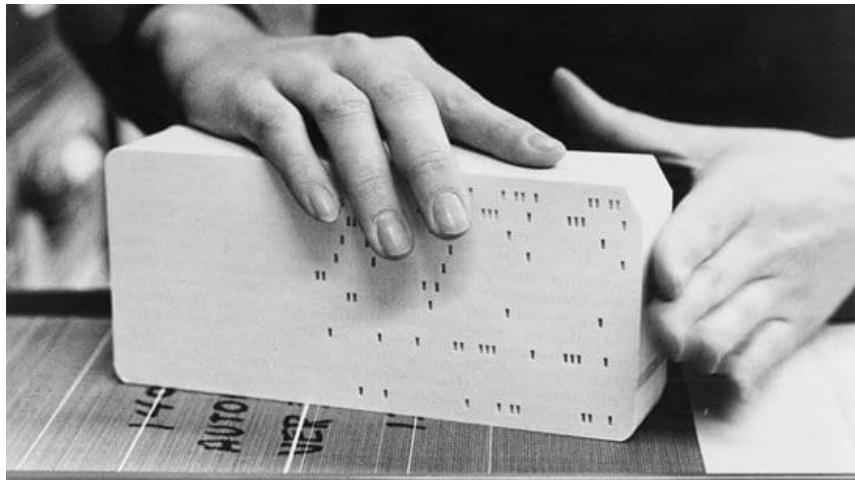
System/360 o S/360, uno de los mayores éxitos de IBM (1961)

El **software**, el conjunto de instrucciones que dirige al hardware, inicialmente requería manipulación física directa del hardware para programar una máquina. Los programas se implementaban a través de la configuración manual de interruptores o el uso de tarjetas perforadas. Esta etapa subraya la importancia de la interacción física en la programación temprana. Sin embargo, con la invención de lenguajes de programación como FORTRAN y COBOL en los años 50, los programadores comenzaron a escribir instrucciones de manera más abstracta, liberándose de las limitaciones del hardware específico. Este cambio fue un punto de inflexión, permitiendo una programación más flexible y sofisticada, y allanando el camino para la programación moderna.



Buenos Aires aprende

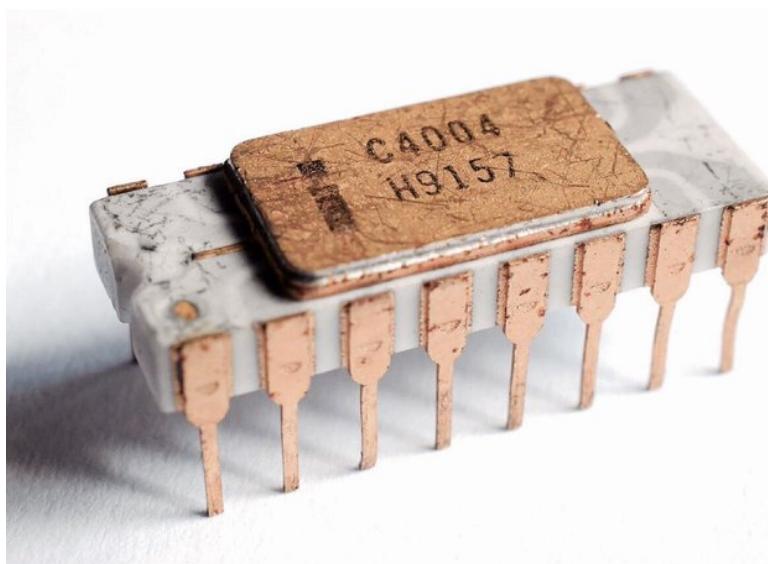
Agencia de Habilidades para el Futuro



Las tarjetas perforadas se usaron para el almacenamiento de rutinas, programas y datos.

La evolución tecnológica llevó a una distinción más clara entre hardware y software. El hardware avanzó hacia la miniaturización y eficiencia, con desarrollos clave como el microprocesador en los años 70, que fueron fundamentales para la creación de las computadoras personales. Este avance no solo democratizó el acceso a la computación, sino que también fomentó la innovación en software. Paralelamente, el software se volvió más complejo y capaz, expandiendo sus funcionalidades más allá de simples cálculos numéricos a aplicaciones gráficas avanzadas, sistemas operativos y redes. Este crecimiento destacó la necesidad de software que pudiera aprovechar plenamente el potencial del hardware avanzado, conduciendo a una interdependencia cada vez mayor entre ambos.

La relación simbiótica entre hardware y software es fundamental para que las computadoras modernas realicen una amplia variedad de tareas, desde el procesamiento de texto hasta gráficos 3D y conectividad global. Para los programadores principiantes es esencial comprender esta dinámica. Esto no solo mejora su habilidad para escribir código más eficiente y resolver problemas de manera efectiva, sino que también les ofrece una perspectiva más completa sobre cómo sus desarrollos interactúan con el mundo físico. Este entendimiento es crucial en una era donde la integración de hardware y software forma la base de innovaciones como la inteligencia artificial, la realidad aumentada y la Internet de las Cosas (IoT), subrayando la importancia de una comprensión holística de la informática.



Primer microprocesador en formato chip: el Intel 4004 de 4 bits (1971)

Definición y objetivo de la programación

Definición y objetivo de la programación.

La programación es un proceso fascinante y esencial en el mundo de la tecnología, definida como la práctica de escribir instrucciones para que un computador realice tareas específicas. Aunque a primera vista parece ser simplemente la escritura de código, la programación es mucho más. Representa una manera única de pensar y abordar problemas, transformando ideas abstractas en aplicaciones concretas y funcionales. La programación no solo se trata de conocer lenguajes de programación y sintaxis; implica también un enfoque lógico y creativo para solucionar problemas, optimizar procesos y crear nuevas posibilidades dentro del mundo digital. Es un campo dinámico que requiere una constante actualización de conocimientos y adaptación a nuevas tecnologías y paradigmas.

Buenos Aires aprende

Agencia de Habilidades para el Futuro




```

42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERUSER_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```

Código fuente de un programa informático.

El **objetivo principal** de la programación es desarrollar programas que lleven a cabo tareas específicas. Estas tareas pueden variar desde simples operaciones matemáticas hasta la gestión de complejos sistemas operativos y aplicaciones avanzadas. La programación es fundamental en la creación de software que impulsa innumerables aspectos de la vida moderna, como la gestión de datos, la comunicación, el entretenimiento y la automatización de procesos industriales. Programar es, en esencia, dar vida a las máquinas, permitiéndoles realizar actividades que facilitan, enriquecen y transforman nuestras vidas cotidianas. La programación ha sido la fuerza motriz detrás de innovaciones significativas en diversos campos, incluyendo la medicina, la educación, la ingeniería y las finanzas, demostrando su versatilidad y su impacto transformador en la sociedad.

Lenguajes de Programación: ¿Qué son y para qué sirven?



Algunos de los lenguajes más populares.



Lenguajes de Programación: ¿Qué son y para qué sirven?

Los lenguajes de programación son herramientas fundamentales en el desarrollo de software, actuando como intermediarios entre los programadores y las computadoras. Estos lenguajes permiten a los programadores escribir instrucciones de manera comprensible, que luego son convertidas en código que la computadora puede ejecutar. Existen numerosos lenguajes de programación, cada uno con sus propias reglas de sintaxis y aplicaciones particulares. Estos lenguajes son diseñados para facilitar la creación de programas eficientes y efectivos, abarcando desde operaciones básicas hasta complejas aplicaciones. Algunos lenguajes están orientados a tareas específicas, mientras que otros son más generales. Su elección depende de varios factores, incluyendo la naturaleza del proyecto, el entorno de desarrollo, la eficiencia requerida, y la experiencia del programador.

Ejemplos de Lenguajes de Programación y sus usos

Ejemplos de Lenguajes de Programación y sus usos:

Cada lenguaje de programación tiene sus propias fortalezas y se elige en función de las necesidades específicas del proyecto en desarrollo.



Python es reconocido por su simplicidad y versatilidad, lo que lo convierte en un lenguaje ideal para aquellos que se inician en la programación. Su sintaxis clara y directa, junto con una extensa biblioteca de recursos, facilita el desarrollo rápido de aplicaciones complejas. Python se utiliza en una amplia gama de aplicaciones, desde el desarrollo web hasta la ciencia de datos y la inteligencia artificial, ofreciendo una gran flexibilidad para abordar diversos proyectos.



Java, por otro lado, es un lenguaje ampliamente utilizado en el desarrollo de aplicaciones empresariales y sistemas móviles, especialmente en la plataforma Android. Es valorado por su robustez y portabilidad, así como por su capacidad para manejar grandes cantidades de datos, lo que lo hace ideal para proyectos a gran escala. La fiabilidad y el rendimiento de Java lo convierten en una opción preferida para aplicaciones que requieren una base sólida y escalable.



Los lenguajes C y C++ son conocidos por su eficiencia y su capacidad para interactuar de manera cercana con el hardware. Estos lenguajes se utilizan comúnmente en el desarrollo de sistemas operativos, juegos y otras aplicaciones donde el rendimiento es crítico. La gestión detallada de los recursos del sistema que permiten C y C++ es esencial en situaciones donde la eficiencia y la velocidad son prioritarias.

Existen cientos de lenguajes de programación más, algunos muy populares como JavaScript o los mencionados anteriormente, y otros prácticamente desconocidos. Pero todos cumplen la misma función: permitir al programador crear programas que serán ejecutados por una computadora.



Algoritmo

Algoritmo.

Un **algoritmo** es un término central en el mundo de la programación y la informática, esencialmente una secuencia de instrucciones o reglas claramente definidas destinadas a realizar una tarea específica o resolver un problema. Lo que distingue a un algoritmo es su naturaleza estructurada y lógica, permitiendo su ejecución paso a paso. La eficacia de un algoritmo no solo reside en su capacidad para lograr un objetivo, sino también en la eficiencia y claridad con la que lleva a cabo la tarea. Los algoritmos son fundamentales en la informática porque proporcionan la base lógica para que las computadoras procesen datos y realicen cálculos. Su universalidad reside en que pueden ser aplicados en diversos contextos y con diferentes tipos de datos, lo que les permite ser una herramienta poderosa en una amplia gama de aplicaciones, desde la resolución de problemas matemáticos simples hasta la conducción de investigaciones complejas y el desarrollo de software avanzado.

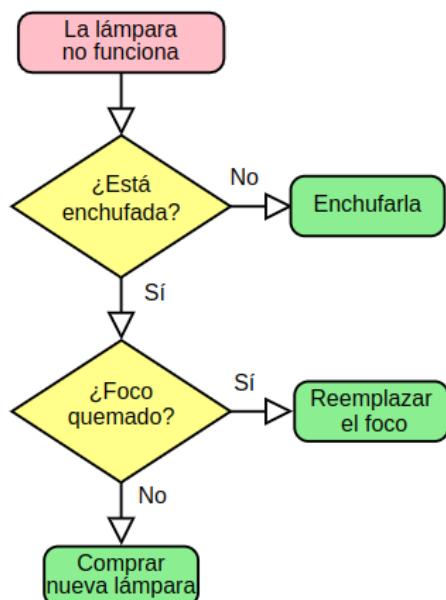


Diagrama de flujo -una de las maneras de representar un algoritmo- para solucionar el motivo por el cuál una lámpara no funciona.



La belleza y la fuerza de un algoritmo radica en su capacidad de ser adaptado y aplicado en diferentes contextos, manteniendo su eficacia y eficiencia. Un algoritmo bien diseñado puede ser utilizado en múltiples situaciones, con distintos conjuntos de datos, y aún así proporcionar resultados consistentes y fiables. Esta adaptabilidad es crucial en campos como la ciencia de datos, la inteligencia artificial, la optimización de motores de búsqueda y el desarrollo de software, donde los algoritmos forman la columna vertebral de las operaciones y decisiones. Un algoritmo efectivo puede significar la diferencia entre un proceso que se ejecuta en segundos y uno que tarda horas, lo que subraya la importancia de la optimización algorítmica en la informática. Además, la comprensión y aplicación de algoritmos es una habilidad valiosa para los programadores y científicos de datos, ya que les permite abordar problemas complejos de manera estructurada y eficiente, llevando a soluciones innovadoras y efectivas en el ámbito tecnológico.

Definición y características.

Definición y características.

Un algoritmo efectivo es una herramienta vital en la programación y la informática, distinguido por una serie de características clave. La primera y más importante es la claridad y la precisión. Un algoritmo debe ser lo suficientemente detallado para evitar cualquier ambigüedad, asegurando que cada paso sea comprensible y ejecutable sin dejar espacio para interpretaciones erróneas. Otra característica crucial es que debe ser finito. Esto significa que, después de un número específico de pasos, el algoritmo debe llegar a una conclusión o solución, evitando bucles infinitos o procesos sin fin. Además, cada paso del algoritmo y su secuencia deben estar claramente definidos y bien estructurados. El orden de los pasos es fundamental, ya que dicta la manera en que se ejecutan las instrucciones para alcanzar el resultado deseado de manera eficiente. Estas características aseguran que los algoritmos sean precisos, confiables y efectivos en su aplicación.

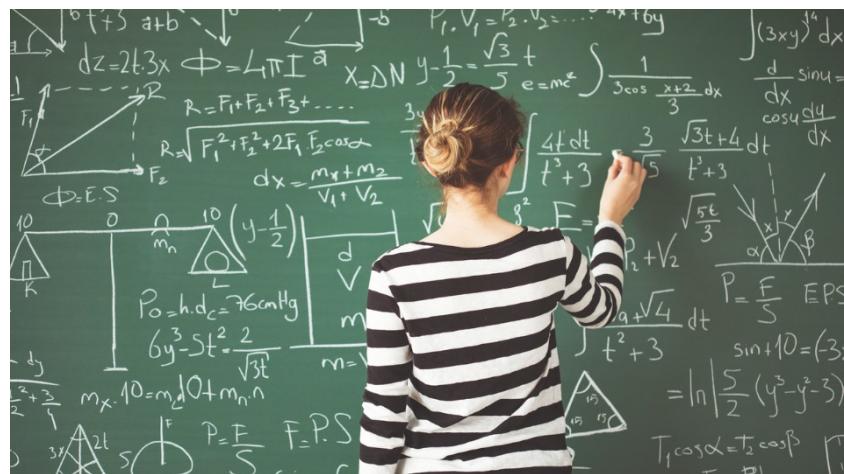
Estas características hacen de los algoritmos herramientas extremadamente poderosas no solo en la programación, sino también en muchas otras áreas de la ciencia y la tecnología. Permiten a programadores y científicos de la computación estructurar sus ideas de manera lógica y transformarlas en programas funcionales y eficientes. En la ciencia de datos, por ejemplo, los algoritmos son fundamentales para analizar grandes conjuntos de datos y

Buenos Aires aprende

Agencia de Habilidades para el Futuro



extraer información significativa. En la inteligencia artificial, los algoritmos permiten a las máquinas aprender de los datos y tomar decisiones autónomas. Además, en el desarrollo de software, un algoritmo bien diseñado es la clave para garantizar que una aplicación funcione de manera óptima. Esta capacidad de los algoritmos para organizar y procesar información de manera efectiva es lo que los convierte en un componente esencial de la tecnología moderna, impulsando la innovación y el progreso en múltiples campos.



Los algoritmos están presentes en prácticamente todos los campos.

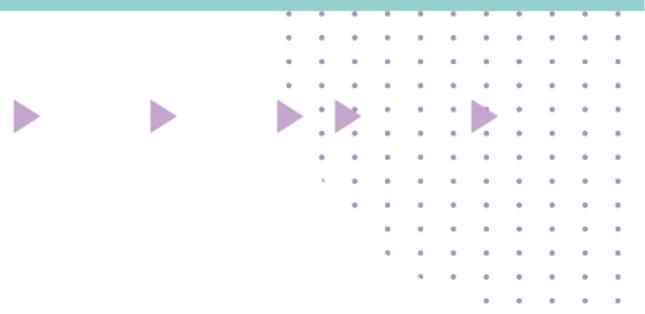
Entrada, proceso y salida.

Entrada, proceso y salida

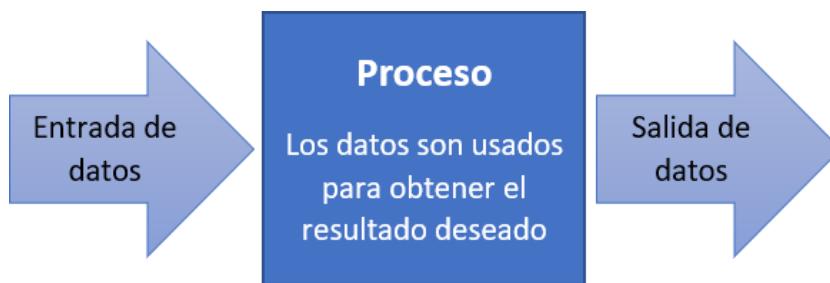
Los algoritmos están compuestos por tres elementos fundamentales: entrada, proceso y salida.

Entrada: Se refiere a los datos que el algoritmo recibe. Estos pueden ser números, texto, o cualquier otro tipo de información que será utilizada en el proceso de cálculo o toma de decisiones.

Proceso: Es el conjunto de operaciones o instrucciones que el algoritmo ejecuta con los datos de entrada. Aquí es donde se lleva a cabo la lógica del algoritmo, realizando cálculos o tomando decisiones basadas en los datos de entrada.



Salida: Son los resultados que el algoritmo produce después de realizar sus procesos. Estos resultados pueden variar dependiendo del tipo de algoritmo y de los datos de entrada que se le proporcionen.



Entender estos componentes es crucial para diseñar algoritmos efectivos y eficientes, ya que proporcionan una estructura clara sobre cómo se deben organizar y ejecutar las tareas.

Claridad, finitud, definición, orden.

Claridad, finitud, definición, orden.

Para que un algoritmo sea efectivo y funcional, es esencial que adhiera a cuatro principios fundamentales: claridad, finitud, definición y orden. Estos principios son los pilares que aseguran la eficiencia y la efectividad de cualquier algoritmo.

Claridad: La claridad es primordial en un algoritmo. Cada instrucción debe ser precisa y fácil de entender, eliminando cualquier posibilidad de ambigüedad. Esto asegura que cualquiera que lea o implemente el algoritmo tenga una comprensión exacta de lo que debe hacerse en cada paso. La claridad no solo facilita la implementación y el mantenimiento del algoritmo, sino que también hace que la depuración y la revisión sean más sencillas.

Finitud: La finitud implica que el algoritmo debe tener un final definido. Después de un número determinado de pasos, el algoritmo debe concluir su ejecución. Esto evita bucles infinitos y asegura que el algoritmo produzca un resultado en un tiempo razonable. La finitud es crucial para la eficacia del algoritmo, garantizando que cumpla su propósito de manera oportuna.



Definición: Cada paso del algoritmo debe estar claramente definido, sin dejar lugar a dudas sobre lo que se debe hacer en cada momento. Una definición precisa de cada paso ayuda a prevenir errores y malentendidos durante la ejecución del algoritmo, lo que es esencial para su correcto funcionamiento y fiabilidad.

Orden: El orden en el que se ejecutan los pasos en un algoritmo es vital. La secuencia dicta el funcionamiento del algoritmo y, por lo tanto, su resultado final. Un cambio en el orden de los pasos puede alterar significativamente los resultados, lo que hace que el establecimiento de un orden lógico y eficiente sea un aspecto crítico en el diseño del algoritmo.

Al seguir estos principios, las programadoras y programadores pueden garantizar que sus algoritmos sean robustos, confiables y eficientes. Estos fundamentos no sólo aseguran que el algoritmo funcione según lo previsto, sino que también lo hacen útil y práctico para la resolución de problemas en una variedad de contextos. La adhesión a estos principios es lo que diferencia a los algoritmos bien diseñados de aquellos que son propensos a errores y fallas, y es un aspecto crucial en la creación de soluciones tecnológicas efectivas y fiables.

Representación de Algoritmos

Representación de Algoritmos

La representación de algoritmos se puede realizar de diversas maneras: cada una adecuada para diferentes etapas del desarrollo o para diferentes públicos. Las tres formas más comunes son los diagramas de flujo, el pseudocódigo y el código fuente.

Diagramas de Flujo: Son representaciones gráficas de los pasos de un algoritmo. Utilizan símbolos estándar para representar diferentes tipos de instrucciones o acciones, como operaciones de entrada/salida, decisiones y procesos. Son especialmente útiles para visualizar el flujo lógico de un algoritmo y son ideales para explicar procesos complejos de manera sencilla y clara.

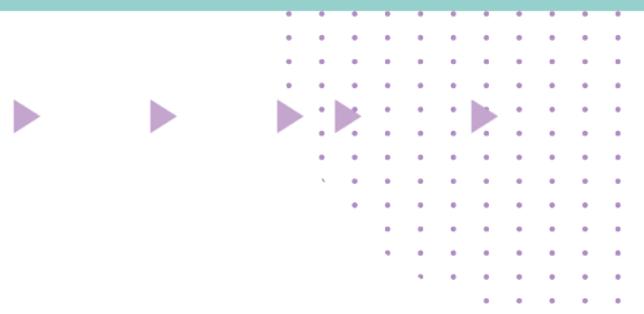


Diagrama de flujo.

Pseudocódigo: Es una forma de escribir algoritmos que utiliza un lenguaje cercano al natural con algunas estructuras de lenguajes de programación. No sigue la sintaxis estricta de ningún lenguaje específico, pero ayuda a planificar y estructurar el algoritmo de manera clara y organizada. El *pseudocódigo* es un paso intermedio útil entre la formulación del problema y la escritura del código fuente que luego interpretará el computador.

```

1  Algoritmo PAR_IMPAR
2      Escribir 'Por favor, teclea un número'
3      Leer numero
4      Si numero MOD 2=0 Entonces
5          Escribir 'El número, ',numero,' es par'
6      SiNo
7          Escribir 'El número, ',numero,' es impar'
8      FinSi
9  FinAlgoritmo

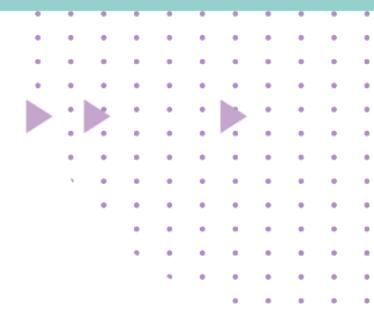
```

Algoritmo escrito en pseudocódigo



Buenos Aires aprende

Agencia de Habilidades para el Futuro



Código Fuente: Es la representación de un algoritmo en un lenguaje de programación específico. Aquí, el algoritmo se escribe siguiendo las reglas sintácticas y semánticas del lenguaje elegido, como PSeInt, Python, Java o C++, entre otros. El código fuente es lo que finalmente se ejecuta en una computadora.

```
107     @routes.route('/login', methods=['GET', 'POST'])
108     def login():
109         if request.method == 'POST':
110             nombre = request.form['username']
111             password = request.form['password']
112             user = Usuario.query.filter_by(nombre=nombre).first()
113             if user and check_password_hash(user.password, password):
114                 session['username'] = user.nombre
115                 session['correo'] = user.correo
116                 session['id_usuario'] = user.id_usuario
117                 return redirect(url_for('seleccionar_fecha'))
118             else:
119                 return render_template('login.html', error='Usuario')
120         return render_template('login.html')
121     
```

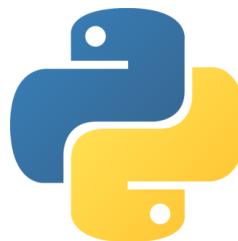
Código fuente escrito en Python.

Cada una de estas representaciones tiene su propósito y su público objetivo. Mientras que los diagramas de flujo y el pseudocódigo son excelentes para la enseñanza y la planificación, el código fuente es necesario para la implementación efectiva de los algoritmos en aplicaciones reales.



Introducción al lenguaje Python

Introducción al lenguaje Python



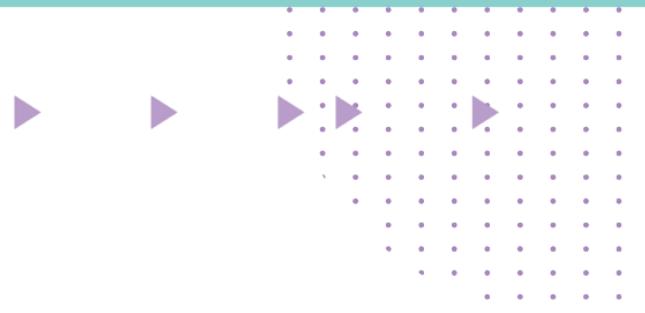
Python es un lenguaje de programación interpretado de alto nivel, caracterizado por su énfasis en la simplicidad y legibilidad del código. Creado a finales de 1989 por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI) en los Países Bajos, Python se concibió como un sucesor de un lenguaje anterior llamado ABC, capaz de manejar excepciones y de interactuar con el sistema operativo Amoeba.

La filosofía detrás de Python se centra en dos principios fundamentales: **la legibilidad del código y la simplicidad tanto en la escritura como en la lectura**. Estos principios se reflejan en el diseño del lenguaje, que promueve estructuras de código claras y un estilo de programación que **prioriza la facilidad de comprensión sobre la brevedad del código**. Esta filosofía se encapsula en el *"Zen de Python"*, una colección de 19 aforismos que guían el diseño del lenguaje.

Características claves de Python

Características claves de Python

Python se destaca por una serie de características que lo hacen accesible y poderoso tanto para quienes recién comienzan a programar como para aquellas personas con experiencia. Una de las características más notables es su sintaxis clara y legible que se asemeja al lenguaje natural. Esto permite escribir código de manera intuitiva y comprensible, facilitando la tarea de desarrollar aplicaciones y sistemas sin enfrentarse a la complejidad que a menudo se encuentra en otros lenguajes.



Evolución y popularidad

Evolución y popularidad

Desde su creación, Python ha experimentado un crecimiento constante en popularidad, impulsado tanto por su facilidad de aprendizaje como por su poderosa funcionalidad. Su uso abarca una amplia gama de aplicaciones, desde desarrollo web y automatización de tareas hasta ciencia de datos, aprendizaje automático e inteligencia artificial. La adopción de Python en la comunidad científica y de investigación ha sido particularmente notable, gracias a su simplicidad y la disponibilidad de numerosas bibliotecas especializadas.

La comunidad de Python es una de las más activas y colaborativas en el mundo del software libre y de código abierto. Con una amplia gama de conferencias, talleres y foros dedicados a Python, los desarrolladores tienen muchas oportunidades para aprender y contribuir al ecosistema de Python.

Python se distingue por ser un lenguaje que combina eficiencia, claridad y potencia, ofreciendo un equilibrio ideal entre facilidad de aprendizaje y capacidad para realizar tareas complejas. Su diseño orientado a mejorar la experiencia tanto de programadores novatos como experimentados ha llevado a Python a ser uno de los lenguajes de programación más enseñados en las universidades y más empleados en la industria hoy en día.



