



PENSAMIENTO ALGORÍTMICO

DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS

El pensamiento algorítmico es una competencia clave en el desarrollo de habilidades de programación moderna. En esta unidad, se profundiza en los conceptos de diseño de algoritmos eficientes, explorando su implementación en Python y la utilización de IA para optimizar y analizar la eficiencia del código. Se presentarán ejemplos prácticos que permitirán a los estudiantes aplicar la lógica, las estructuras de control y optimizaciones para crear soluciones ágiles y eficaces, aprovechando los recursos tecnológicos actuales.

¡Prepárese para transformar sus habilidades de programación y optimización con técnicas y herramientas de última generación!

INICIAR



TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

Todo el contenido de este curso es propiedad intelectual de la Corporación Tecnológica del Oriente y está protegido por derechos de autor. No puede ser reproducido, distribuido, modificado ni compartido sin su autorización por escrito.

UNIDAD 3. DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS

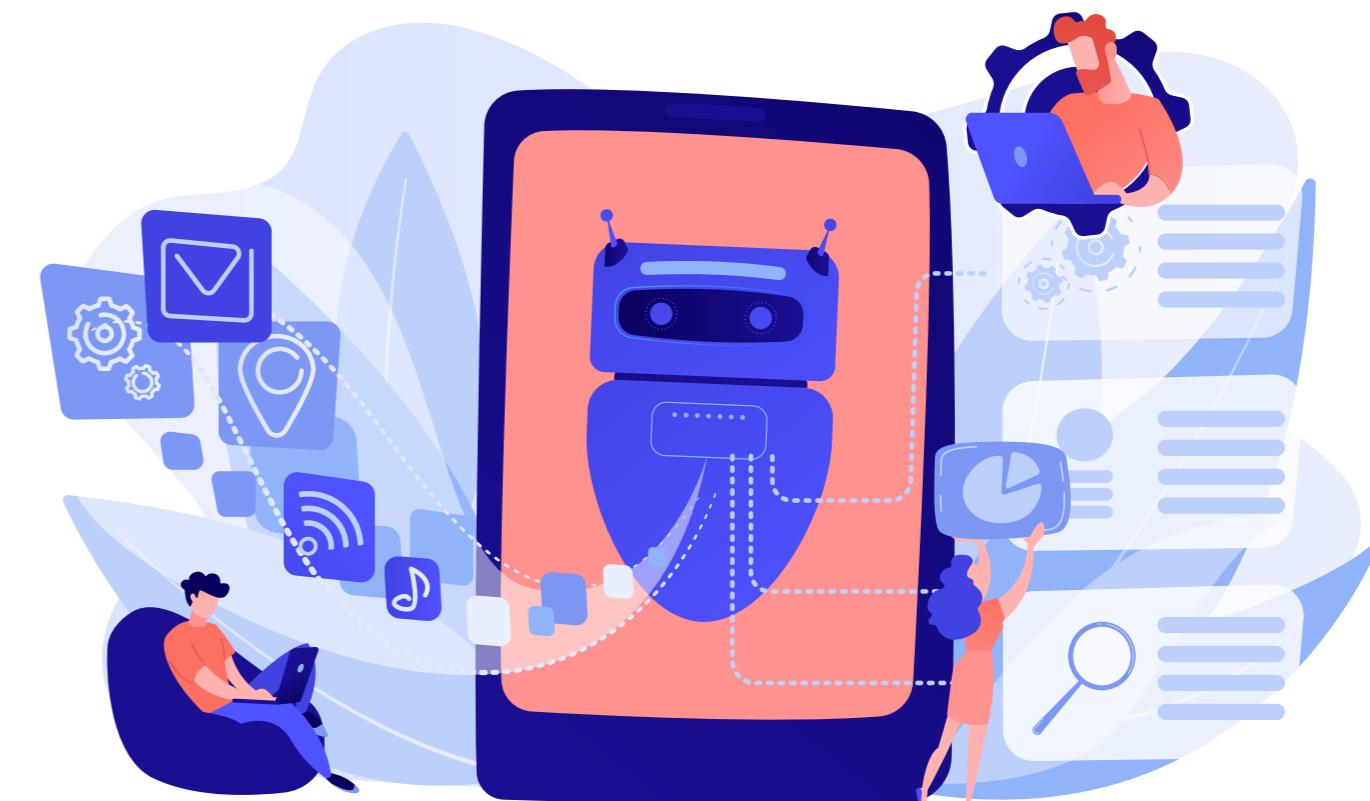
INTRODUCCIÓN



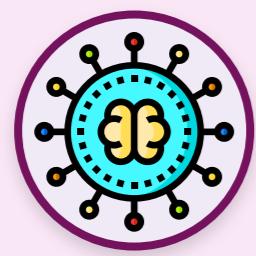
El pensamiento algorítmico es una competencia fundamental en el mundo de la programación moderna, esta unidad está diseñada para desarrollar habilidades fundamentales en el diseño, implementación y análisis de algoritmos eficientes, los estudiantes aprenderán a abordar problemas mediante el pensamiento algorítmico, a implementar soluciones optimizadas utilizando Python como lenguaje principal, aplicando los principios de la lógica y las estructuras básicas de control de flujo; además podrá implementar la IA para la optimización del código.

La temática que se abordará está dividida en tres temas importantes:

- ➡ Conceptos generales en el diseño de algoritmos eficientes.
- ➡ Implementación de algoritmos en lenguaje de programación básico Python con ejemplos.
- ➡ Utilización de IA para la optimización y análisis básicos en la eficiencia algorítmica.



Al finalizar, estará en la capacidad de diseñar algoritmos utilizando pseudocódigo y emplea Python como lenguaje de programación, para finalizar podrá ajustar el código creado usando IA para mejorar su rendimiento y funcionalidad.



Le invitamos a descubrir la importancia de la eficiencia algorítmica a través de esta unidad y a reconocer la importancia del uso de las IA en el alcance de este objetivo.

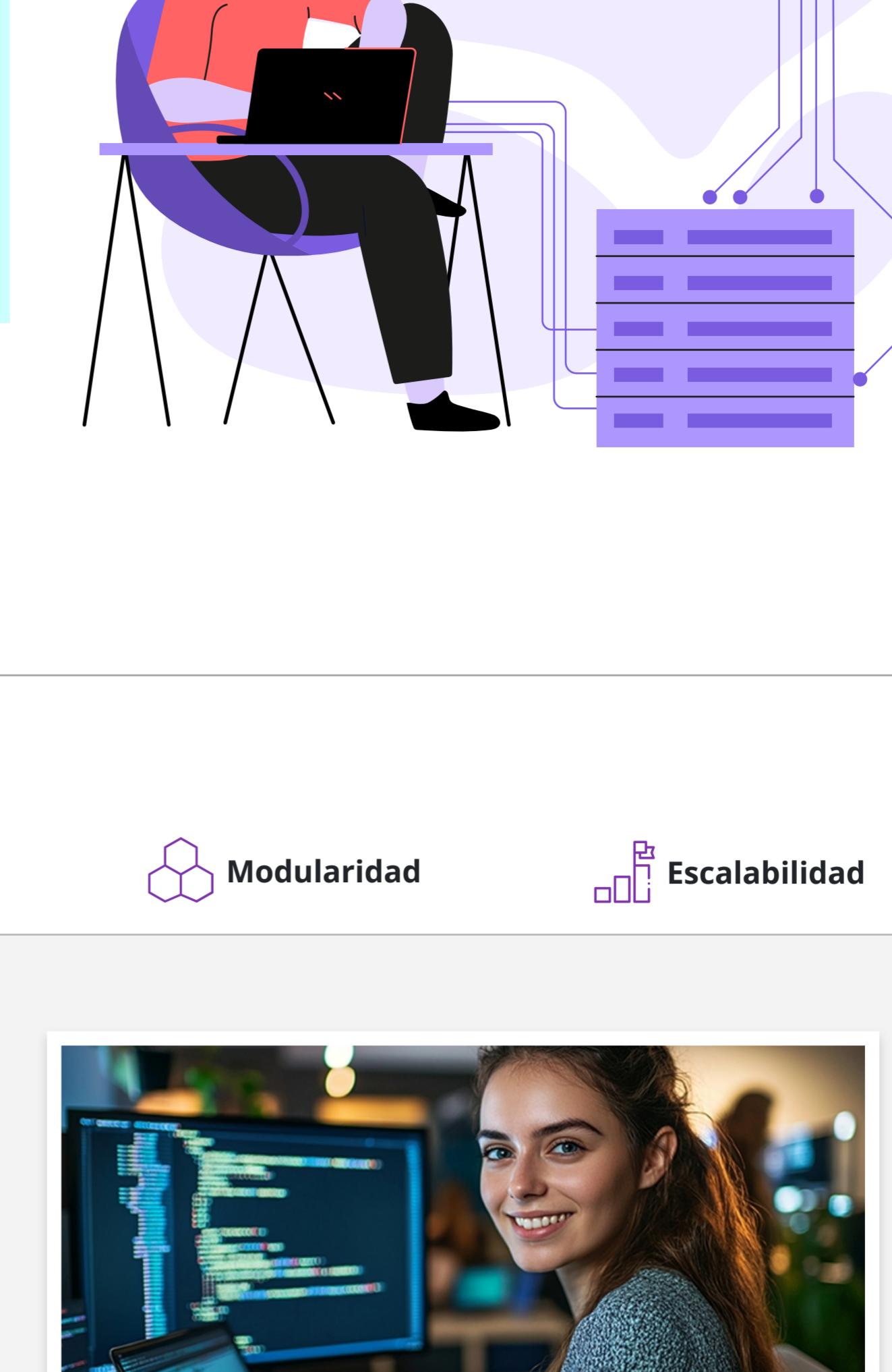
UNIDAD 3. DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS

1. CONCEPTOS GENERALES EN EL DISEÑO DE ALGORITMOS EFICIENTES

Antes de comenzar con el desarrollo de este tema, es importante que repase los conceptos generales de algoritmos; así podrá comprender con mayor grado de precisión cada uno de los ejemplos aquí utilizados, además de proponer soluciones de mejora para cada caso.



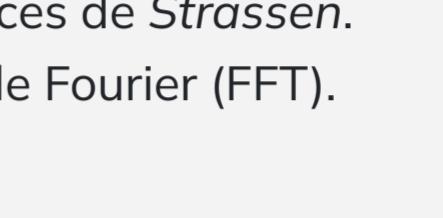
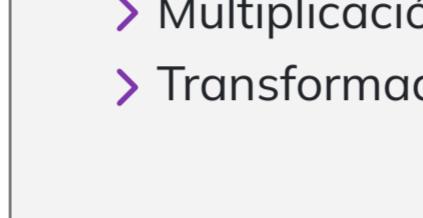
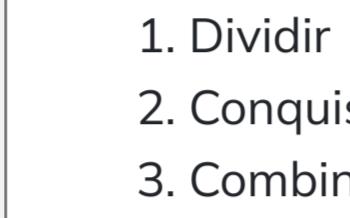
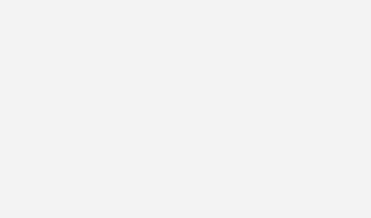
En la construcción de algoritmos se pueden encontrar diferentes enfoques que permiten dar solución a los problemas computacionales. Los paradigmas en el diseño de algoritmos son como un maletín repleto de herramientas, cada una con una finalidad específica; así podrá utilizar según las fortalezas y el caso propuesto alguna de ellas. Como señala Mancilla Herrera (2015), "el dominio de diferentes paradigmas de diseño te permitirá abordar problemas complejos de manera sistemática y eficiente" (p. 67).



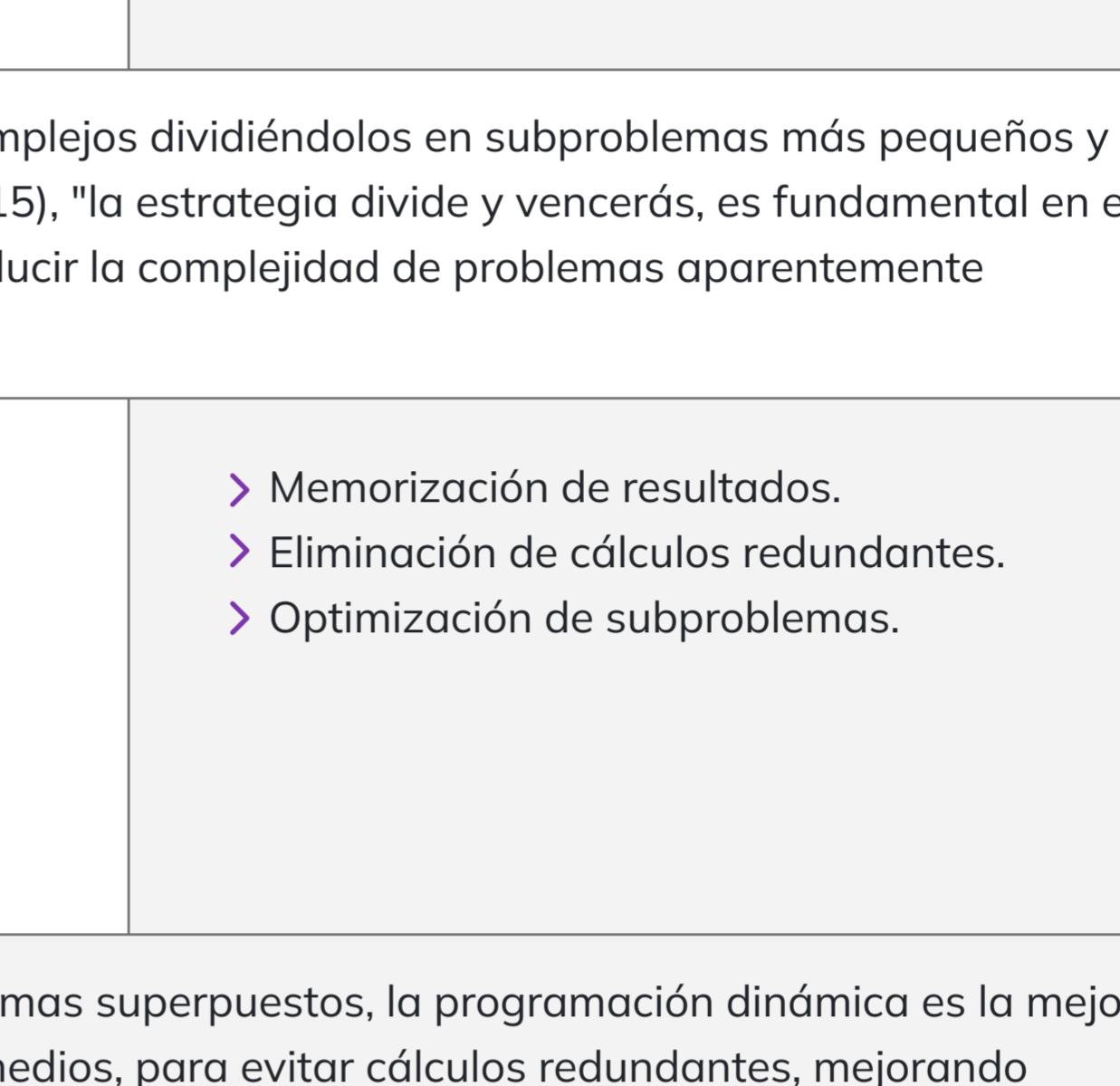
A continuación se darán a conocer los grandes rasgos en los fundamentos del diseño de algoritmos.

1.1 Fundamentos del diseño algorítmico

Los principios que rigen el diseño de un algoritmo son:

**Corrección**

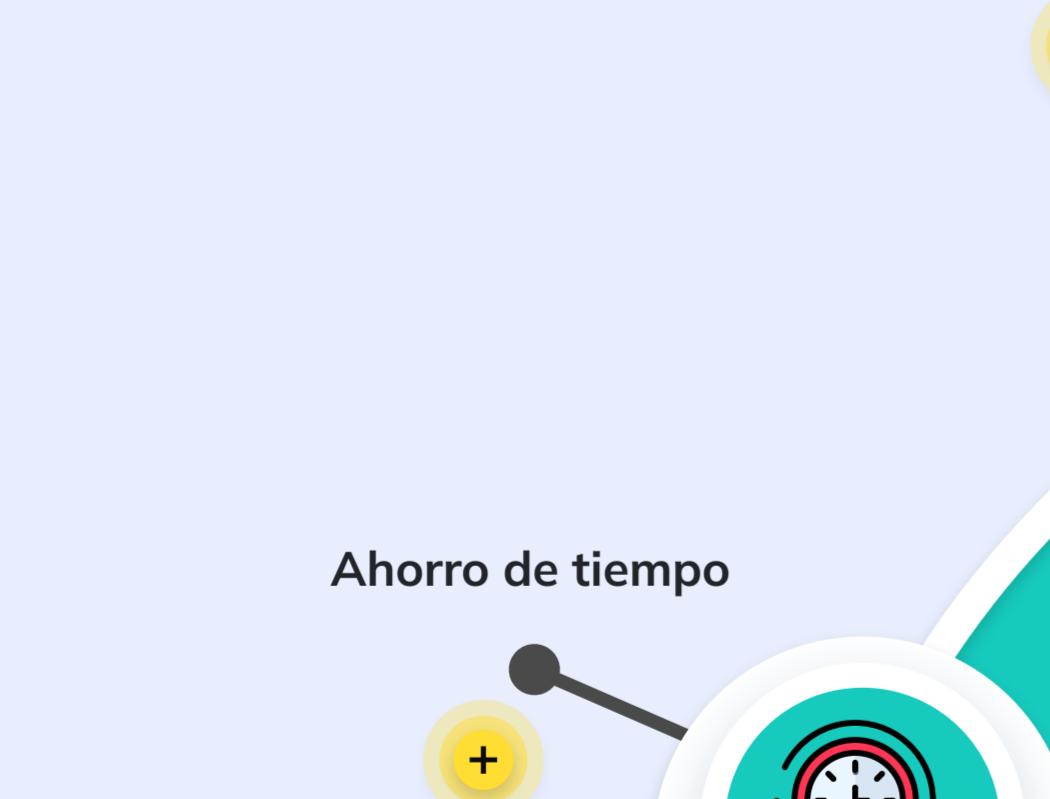
Para todas las entradas de datos válidos se deberá producir la salida correcta.



Algunos de los paradigmas más importantes utilizados a la hora de producir algoritmos son:

Tabla 1 . Paradigmas más utilizados al producir algoritmos

Paradigma	Características	Aplicaciones prácticas
Divide y vencerás	Los pasos que sigue este paradigma son: 1. Dividir 2. Conquistar 3. Combinar	> Ordenamiento (MergeSort, QuickSort). > Multiplicación de matrices de Strassen. > Transformada rápida de Fourier (FFT).
Divide and Conquer	En el paso 1, se descompone el problema en subproblemas; posteriormente, en el paso 2 se resuelven cada uno de esos "subproblemas, usando recursividad"; para finalizar, se utiliza combinar, para unir las soluciones a los subproblemas.	
	Este paradigma permite abordar problemas complejos dividiéndolos en subproblemas más pequeños y manejables. Como señala Mancilla Herrera (2015), "la estrategia divide y vencerás, es fundamental en el diseño de algoritmos eficientes, permitiendo reducir la complejidad de problemas aparentemente intratables" (p. 45).	
Programación dinámica	Se utiliza cuando: > Los problemas se superponen. > Los problemas tienen una subestructura óptima. > Las soluciones de los subproblemas se pueden almacenar.	> Memorización de resultados. > Eliminación de cálculos redundantes. > Optimización de subproblemas.
	Cuando se enfrenta a problemas con subproblemas superpuestos, la programación dinámica es la mejor aliada. Esta técnica almacena resultados intermedios, para evitar cálculos redundantes, mejorando significativamente la eficiencia.	
Algoritmos voraces	Principios de diseño: 1. Selección voraz: elegir la mejor opción disponible. 2. Factibilidad: verificar si la elección mantiene una solución viable. 3. Optimización local: buscar el óptimo en cada paso.	> Problema de la mochila (Knapsack Problem). > Problema de la moneda mínima (Coin Change Problem). > Árboles de expansión mínima (Minimum Spanning Tree). > Algoritmo de Huffman (Compresión de datos). > Problema de selección de actividades.
Greedy	Según Silva Ramírez (2018), "los algoritmos voraces toman decisiones localmente óptimas en cada paso, esperando llegar a una solución globalmente óptima" (p. 145).	
		En situaciones donde se requiera tomar decisiones paso a paso, los algoritmos voraces permiten hacer la mejor elección local en cada momento, buscando alcanzar una solución óptima global.
Vuelta atrás	Este paradigma es esencial para la búsqueda exhaustiva, donde se requiere: > Explorar todas las posibilidades. > Deshacer decisiones que no produzcan soluciones. > Encontrar todas las soluciones posibles.	> Sudoku. > Problema de las N reinas. > Generación de subconjuntos. > Búsqueda de caminos en laberintos. > Resolución de problemas de partición. > Solución de puzzles y juegos.
Backtracking	El backtracking es una herramienta que permite explorar muchas combinaciones posibles, "especialmente cuando no hay un criterio directo para saber, de inmediato, cuál es la mejor opción".	

1.2 Análisis y optimización de eficiencia

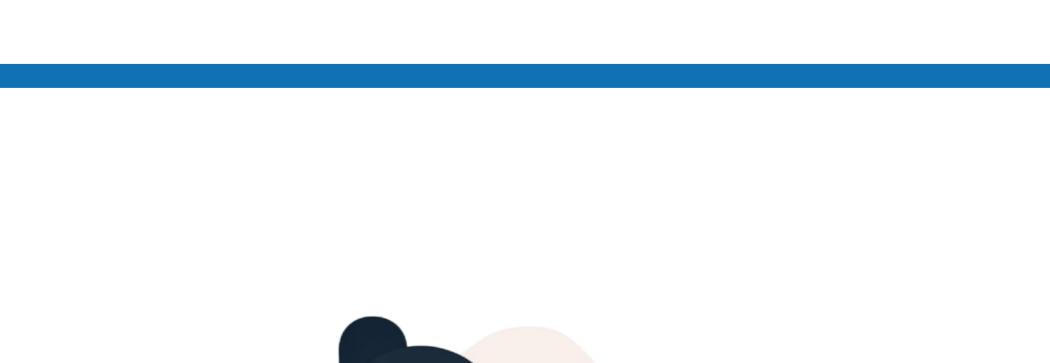
La eficiencia es un aspecto crucial en el desarrollo de algoritmos. Como señala Castillo Romero (2019), "la diferencia entre un algoritmo eficiente y uno ineficiente, puede significar la diferencia entre resolver un problema en segundos o en años" (p. 178). Por su parte, Silva Ramírez (2018), "el análisis de eficiencia no solo nos permite comparar algoritmos, sino que también nos ayuda a predecir su comportamiento, en diferentes escenarios" (p. 78).

Tabla 2. Técnicas de optimización de eficiencia

Métricas de rendimiento	Tiempo de ejecución	> Tiempo real (wall-clock time). > Tiempo de CPU. > Número de operaciones básicas.
Notación asintótica	Uso de memoria	> Memoria estática. > Memoria dinámica. > Pila de llamadas recursivas.
	Notación asintótica	La notación asintótica permite describir el comportamiento del algoritmo sin depender de detalles de implementación específicos.
Técnicas de optimización	Optimización de código	Mejorar partes del código para que sea más eficiente aplicando: > Eliminación de código innecesario. > Simplificar operaciones.
	Optimización de bucles	Se enfoca en la optimización de bucles o ciclos, algunas de las técnicas son: > Reducir el número de iteraciones. > Terminar antes de tiempo el bucle. > Mover operaciones fuera del bucle.



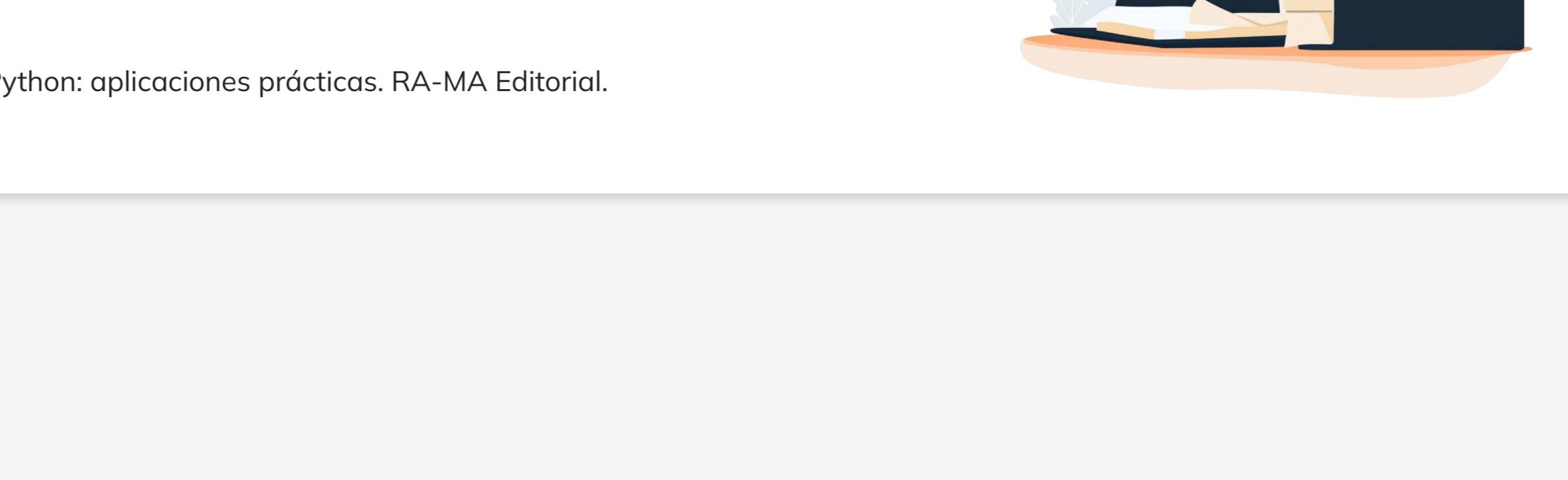
Estas técnicas de optimización de algoritmos, son útiles al trabajar con grandes cantidades de datos o cuando se debe optimizar el programa para que responda más rápido.



La finalidad del concepto de un algoritmo eficiente se centra en el diseño y análisis de soluciones que permitan resolver de manera ágil, y utilizar la menor cantidad de recurso posible, como tiempo de ejecución y memoria, es decir, la eficiencia se traduce en la capacidad del algoritmo de realizar su tarea, utilizando la menor cantidad de medios posibles.

1.3 Importancia de los algoritmos eficientes

A continuación, se resumen algunas de las principales importancias de los algoritmos eficientes:



En resumen, el concepto de algoritmos eficientes, se enfoca en la creación de soluciones que se ejecuten de manera óptima y económica posible. Lo anterior es vital para el desarrollo de software y sistemas de entornos donde los recursos son limitados o se deba priorizar el rendimiento.

Imagine dos algoritmos para ordenar una lista de números. Uno de ellos usa un método simple que compara cada número con los demás (como el *Bubble Sort*), mientras que el otro utiliza un método más optimizado (como el *Merge Sort*). Ambos logran ordenar la lista, pero *Merge Sort* lo hace en mucho menos tiempo para listas grandes, porque tiene una eficiencia mucho mayor, en términos de tiempo de ejecución.

Para practicar y perfeccionar los conceptos adquiridos en este primer tema de la unidad, le invitamos a leer los siguientes libros:

Material complementario

Los invitamos a explorar el material complementario de este curso, en esta sección encontrará recursos que le permitirán profundizar y enriquecer su aprendizaje en los temas tratados en esta unidad.

🔗 Introducción a los algoritmos. Del libro: Capítulo 1: Introducción a los algoritmos Moreno, E. (2012). Grafos: fundamentos y algoritmos. Editorial ebooks Patagonia - J.C. Sáez Editor.

🔗 Mancilla Herrera, A. (2015). Diseño y construcción de algoritmos. Universidad del Norte.

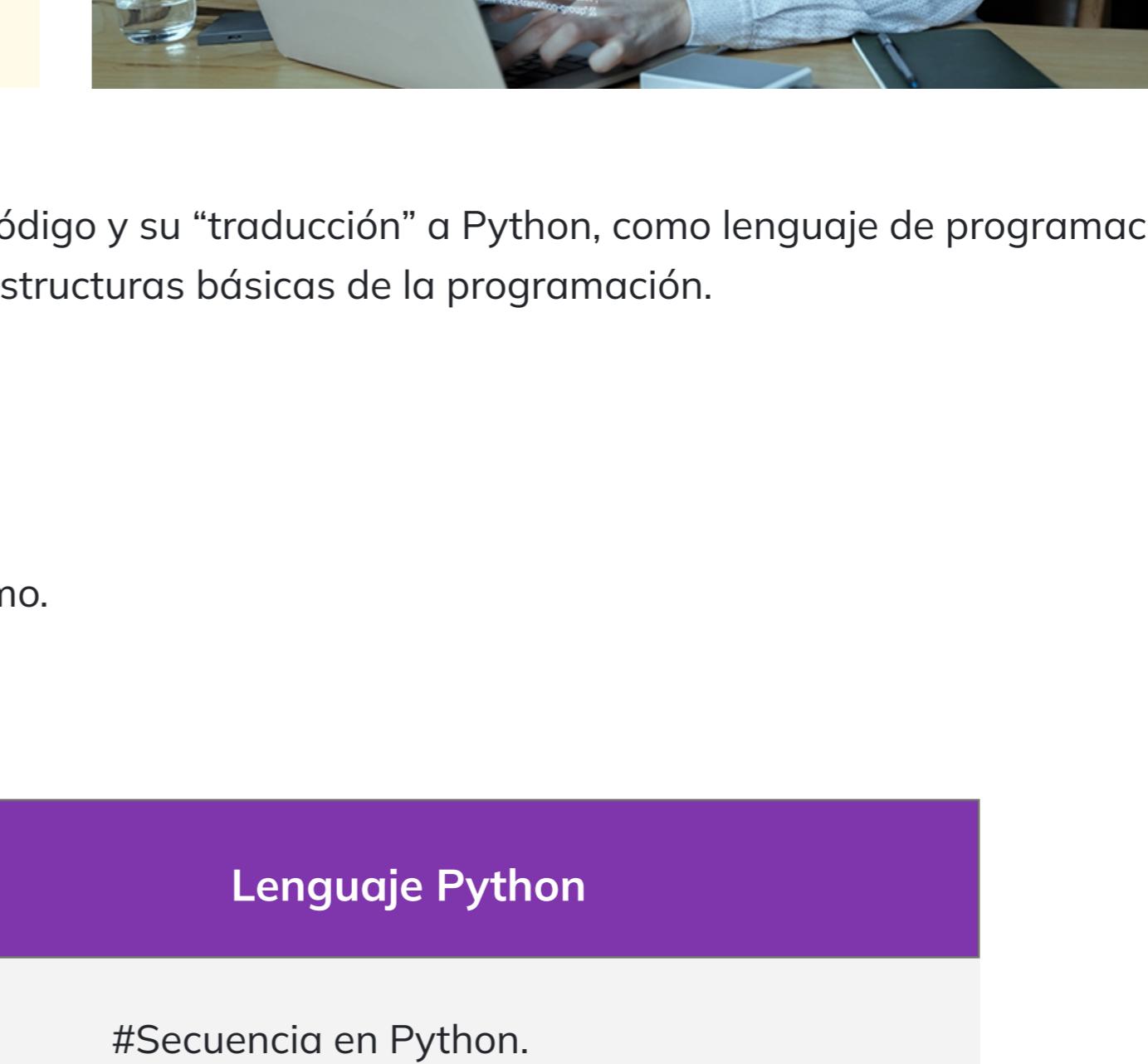
🔗 Nolasco Valenzuela, J. S. (2018). Python: aplicaciones prácticas. RA-MA Editorial.

UNIDAD 3. DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS

2. IMPLEMENTACIÓN DE ALGORITMOS EN LENGUAJE DE PROGRAMACIÓN BÁSICO PYTHON, CON EJEMPLOS

Para adentrarnos en la utilización de lenguajes de programación y para ejemplos prácticos, se utilizará Python por su sencillez y fácil comprensión; es importante recordar algunos conceptos y retomarlos en esta unidad.

El pseudocódigo corresponde a una forma de escribir algoritmos, en la cual se estructura lógicamente un conjunto de instrucciones usando palabras, semejante a como se haría en un lenguaje de programación, pero sin la exigencia y rigidez de este último.



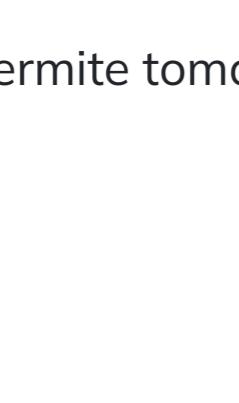
A continuación se realizarán algunos ejemplos en los cuales se presenta el pseudocódigo y su "traducción" a Python, como lenguaje de programación utilizado, para describir el funcionamiento de las **secuencias, condiciones y ciclos**, estructuras básicas de la programación.

Secuencias

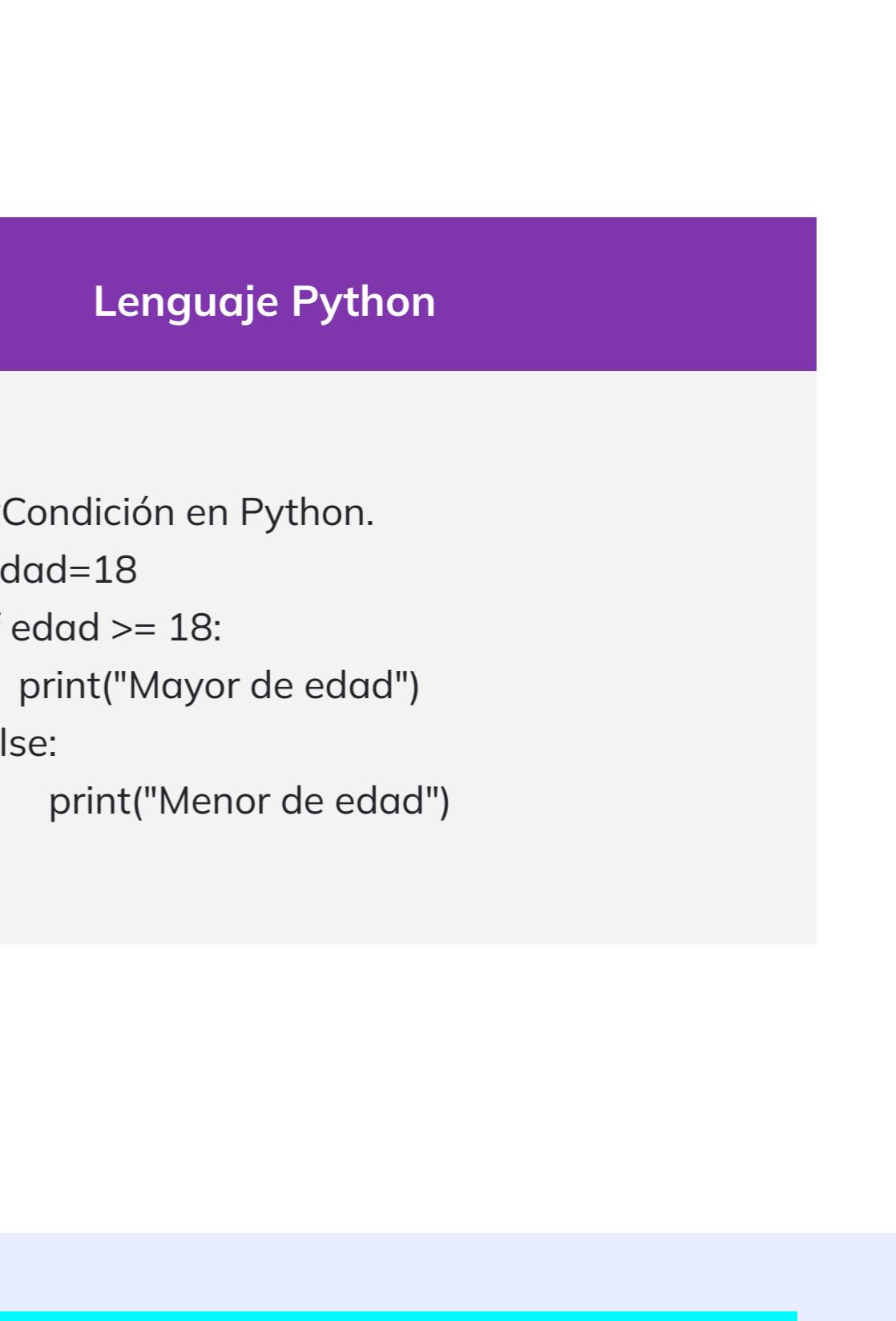
Corresponden al orden lógico en el que debe ejecutarse cada instrucción del algoritmo.

Tabla 3. Ejemplo secuencia en Python

Pseudocódigo	Lenguaje Python
Inicio Definir num1 = 5 Definir num2 = 15 Suma = num1 + num2 Mostrar resultado Fin	#Secuencia en Python. num1=5 num2=15 suma=num1+num2 print("resultado:", suma)



Le invitamos a practicar y reforzar sus conocimientos llevando el código a un compilador en línea, verificando su funcionamiento, cambiando los valores asignados a cada variable y observando los cambios en el resultado; también puede intentar usar otra función como la multiplicación o división, pero recuerde cambiar el nombre de la variable, usando la mnemotecnia, según la operación matemática.



En los ejemplos, se utilizará el compilador online <https://www.programiz.com/python-programming/online-compiler/> como herramienta de programación.

Al copiar el código del ejemplo anterior y pegarlo en el compilador online, el resultado obtenido será el siguiente:

Figura 1. Captura de pantalla compilador online

```

main.py
1 #Secuencia en Python.
2 edad=5
3 num2=15
4 suma=edad+num2
5 print("resultado:", suma)
6

```

Output:
 resultado: 20
 === Code Execution Successful ===

Condición

Permite tomar decisiones dentro de la secuencia de pasos lógicos del algoritmo.

Tabla 4. Ejemplo condición en Python

Pseudocódigo	Lenguaje Python
Inicio Definir edad = 18 si edad >= 18 entonces mostrar "Mayor de edad" Sino Mostrar "Menor de edad" Fin	#Condición en Python. edad=18 if edad >= 18: print("Mayor de edad") else: print("Menor de edad")

Nuevamente copie el código en el compilador online y verifique el funcionamiento.

Figura 2. Captura de pantalla compilador online

```

main.py
1 #Condición en Python.
2 edad=18
3 if edad >= 18:
4     print("Mayor de edad")
5 else:
6     print("Menor de edad")
7

```

Output:
 Mayor de edad
 === Code Execution Successful ===



Le invitamos a cambiar el valor de la variable, edad, por uno menor a 18; ejecute el código y observe el resultado obtenido. Responda: ¿cuál es el objetivo de la decisión dentro del algoritmo? ¿Cómo funciona?

Ciclos

Permiten ejecutar reiterativamente una parte del código, hasta cumplirse una condición; los más usados son: mientras o while y para o for.

Tabla 5. Ejemplo ciclo while en Python

Pseudocódigo	Lenguaje Python
Inicio Definir contador = 1 Mientras contador <= 10 hacer mostrar "contador" incrementar contador en 1 Fin	#Ciclo While en Python. contador=1 while contador <= 10: print(contador) contador+=1

Nuevamente, copie el código en el compilador online y verifique el funcionamiento.

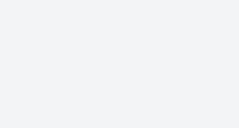
Figura 3. Captura de pantalla compilador online

```

main.py
1 #Ciclo While en Python.
2 contador=1
3 while contador <= 10:
4     print(contador)
5     contador+=1
6

```

Output:
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 === Code Execution Successful ===



Le invitamos a cambiar el valor final del contador y verificar el funcionamiento; después, responda: ¿cuál es la función del ciclo, mientras, dentro del pseudocódigo? ¿Cómo funciona el ciclo, mientras? ¿Qué uso se le puede dar al ciclo, mientras, a la hora de escribir código?

Tabla 6. Ejemplo ciclo For en Python

Pseudocódigo	Lenguaje Python
Inicio Definir num1 = 4 res=0 Para i desde 1 hasta 10 hacer res=num1*i Mostrar "res" Fin	#Ciclo For en Python. num1=4 res=0 for i in range (1, 11): res=num1*i; print(res)

Nuevamente copie el código en el compilador online y verifique el funcionamiento.

Figura 4. Captura de pantalla compilador online

```

main.py
1 #Ciclo For en Python.
2 num1=4
3 res=0
4 for i in range (1, 11):
5     res=num1*i;
6     print(res)
7

```

Output:
 4
 8
 12
 16
 20
 24
 28
 32
 36
 40
 === Code Execution Successful ===

Con el uso del ciclo, for, acabamos de crear la tabla de multiplicar del 4; al revisar el código se puede observar que el incremento se realiza de uno en uno, hasta 11; ¿por qué? ¿qué pasa si se cambia el valor de num1? ¿Qué pasa si varía el rango a (2,11) o (3,11)? ¿Cuál sería la respuesta esperada? ¿Cómo puede ajustar el código para que el resultado presente solamente el valor del factor par, es decir, el resultado de la multiplicación de num1 por 2, 4, 6, 8, 10?

2.1 Ejercicios: combinación de secuencias, condiciones y ciclos

Anexo. Ejercicios.

Los invitamos a explorar el material complementario de este curso, en esta sección encontrará recursos que le permitirán profundizar y enriquecer su aprendizaje en los temas tratados en esta unidad.

► Pacheco, A. (2020). Algoritmos en cuarentena - Ciclos y decisiones (Episodio 2) [video]. YouTube.

🔗 Cuevas Álvarez, A. (2016). Python 3: curso práctico. RA-MA Editorial.

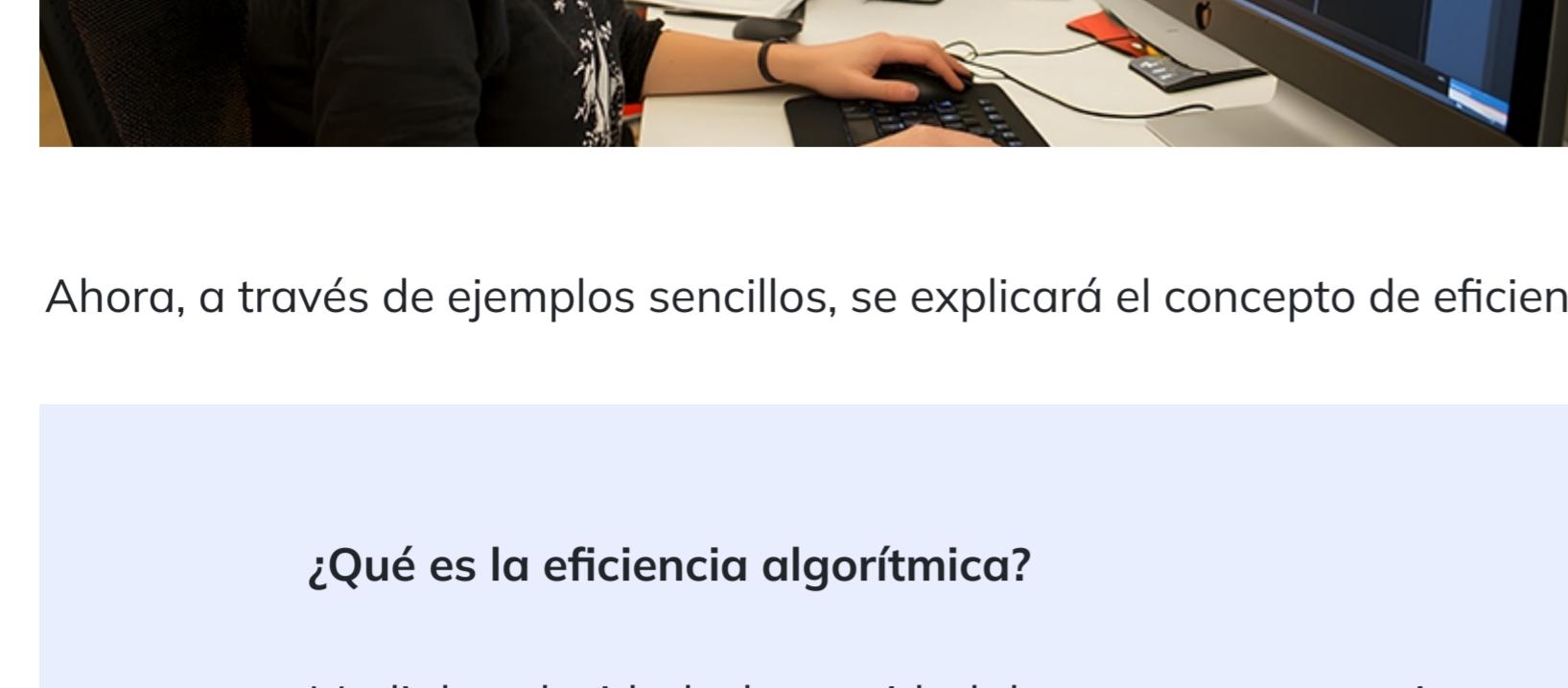
🔗 Hinojosa Gutiérrez, Á. (2015). Python paso a paso. RA-MA Editorial.

UNIDAD 3. DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS

3. UTILIZACIÓN DE IA PARA LA OPTIMIZACIÓN Y ANÁLISIS BÁSICOS EN LA EFICIENCIA ALGORÍTMICA

A continuación, conoceremos cómo se utiliza la IA para la optimización y análisis básicos en la eficiencia.

3.1 Optimización y análisis básicos en la eficiencia algorítmica



Para conocer más sobre el tema, lo invitamos a escuchar el siguiente podcast.



Proceso de descarga de MySQL Server Community y MySQL Workbench

1:00



3:00

Ahora, a través de ejemplos sencillos, se explicará el concepto de eficiencia algorítmica.

¿Qué es la eficiencia algorítmica?

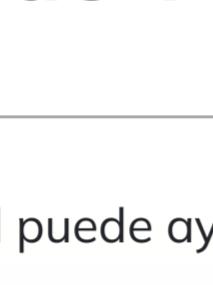
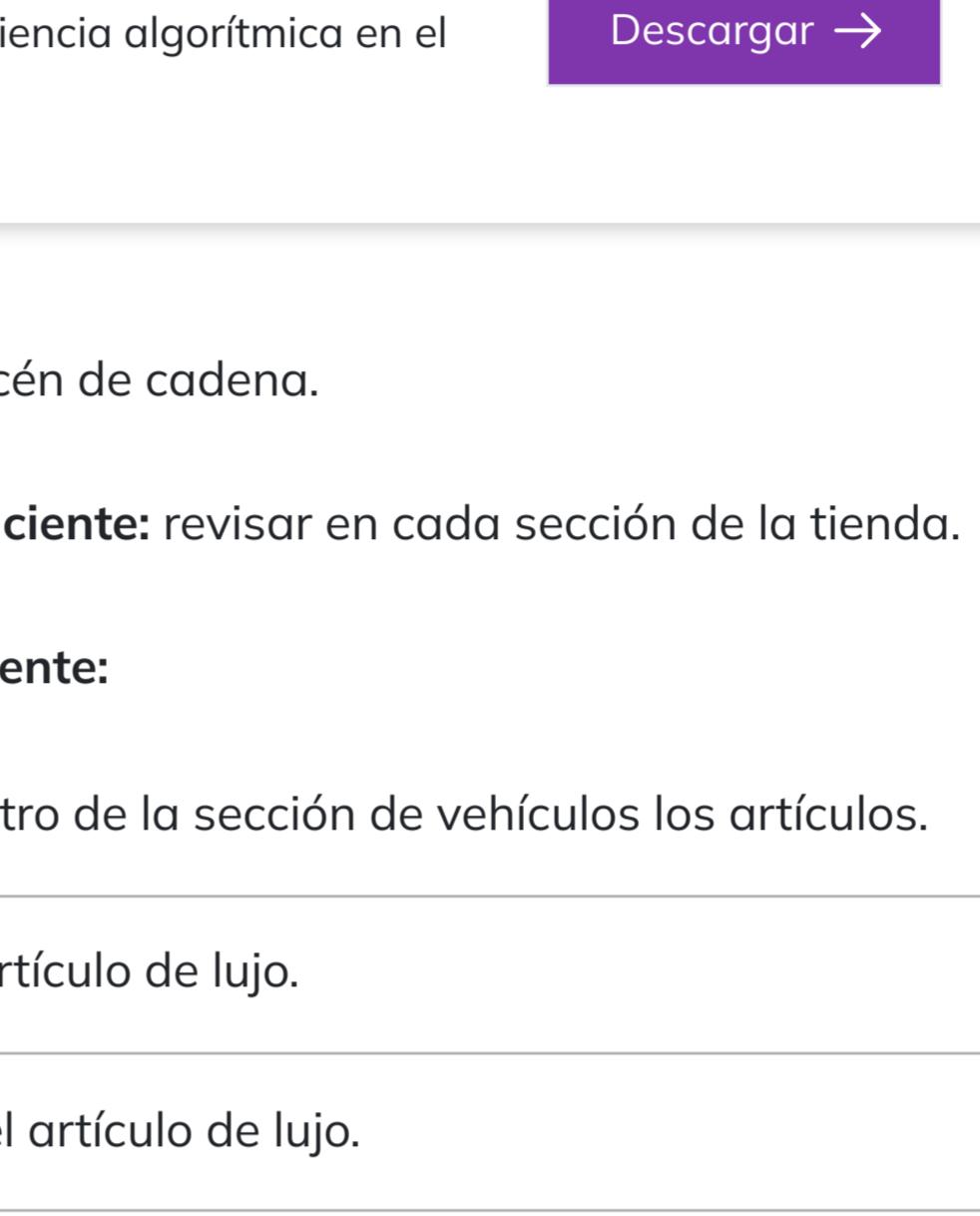
Medir la velocidad y la cantidad de recursos necesarios para que un programa funcione. En términos prácticos, imagine que debe ir al centro de su ciudad y existen dos rutas que le permiten llegar al mismo destino:

- 1 Ruta A: 30 min directo.
- 2 Ruta B: 45 min haciendo transbordo en la estación central.

Responder:

¿Cuál es la ruta más eficiente? ¿Cuál es la ruta que menos recursos consume?

Si en ambas respuestas se seleccionó la ruta A, se ha comprendido la definición de eficiencia algorítmica.

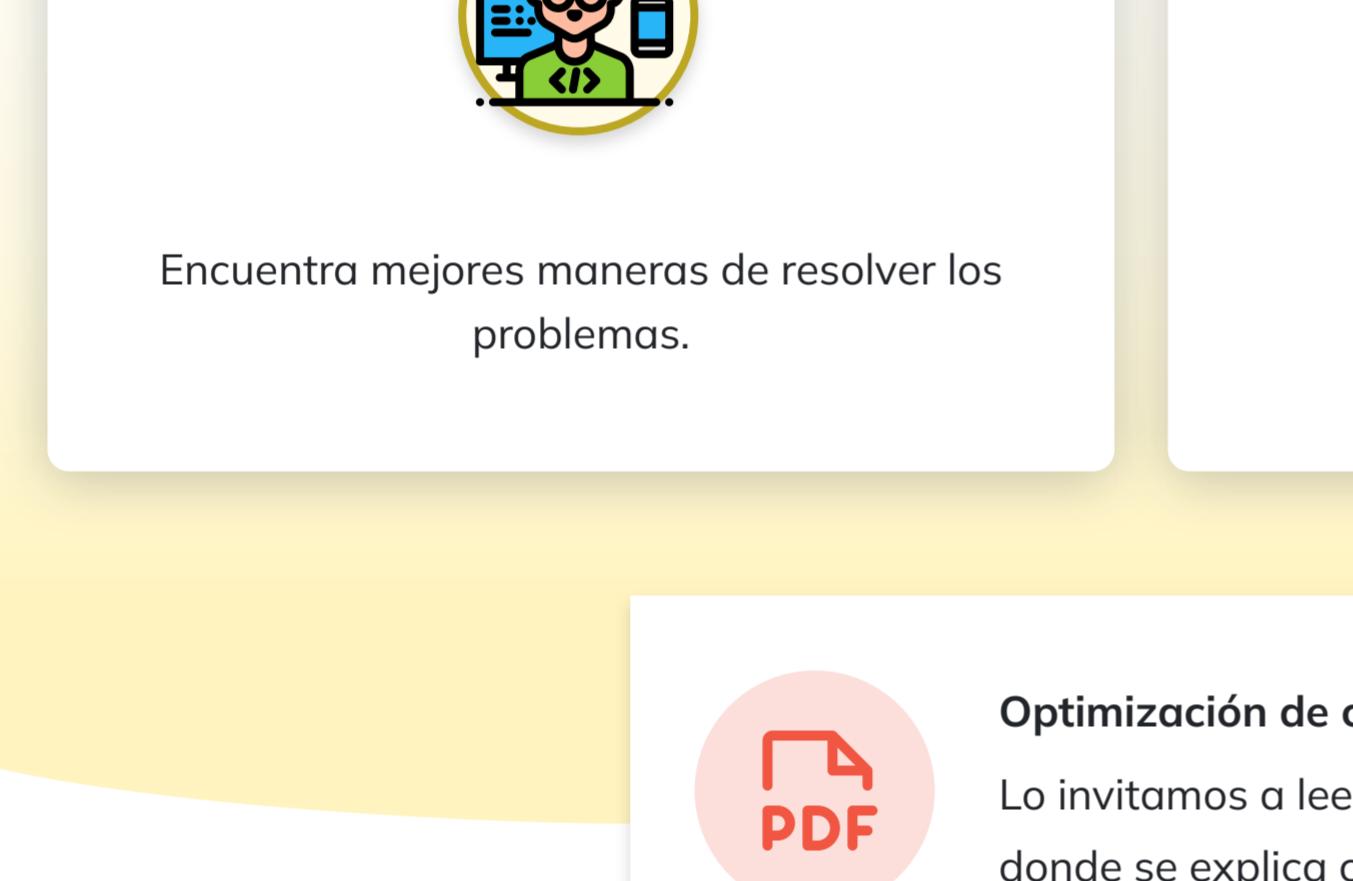


PDF Ejemplo práctico en Python.

Conozca un ejemplo de aplicación de eficiencia algorítmica en el PDF Ejemplo práctico en Python.

[Descargar →](#)

Imagine que se quiere comprar un artículo de lujo para su vehículo en un almacén de cadena.



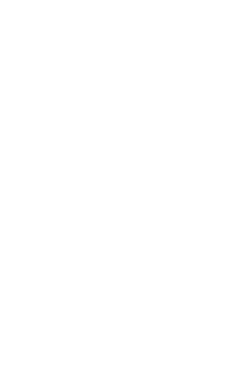
Búsqueda ineficiente: revisar en cada sección de la tienda.

Búsqueda eficiente:

- 1 Buscar dentro de la sección de vehículos los artículos.
- 2 Buscar el artículo de lujo.
- 3 Encontrar el artículo de lujo.

3.2 Utilización de IA para la optimización algorítmica

La IA o Inteligencia Artificial puede ayudar a optimizar algoritmos de la siguiente manera:



Encuentra mejores maneras de resolver los problemas.



Predice la solución más rápido.



Ajusta automáticamente el código para que funcione mejor.

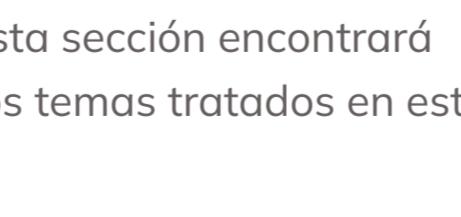


Optimización de códigos sencillos

Lo invitamos a leer el PDF Optimización de códigos sencillos, donde se explica cómo aplicar la optimización de código utilizando dos ejemplos de IA.

[Descargar →](#)

Le invitamos a observar el comportamiento y respuesta entregada por cada IA en los siguientes videos:



Todas las IA tienen una finalidad, dependerá de usted cuál desea usar, según su practicidad, facilidad de uso y tipo de respuesta obtenida. Le invitamos a poner en práctica los temas aprendidos y a usar la IA para mejorar el rendimiento de los algoritmos diseñados para garantizar su eficiencia.

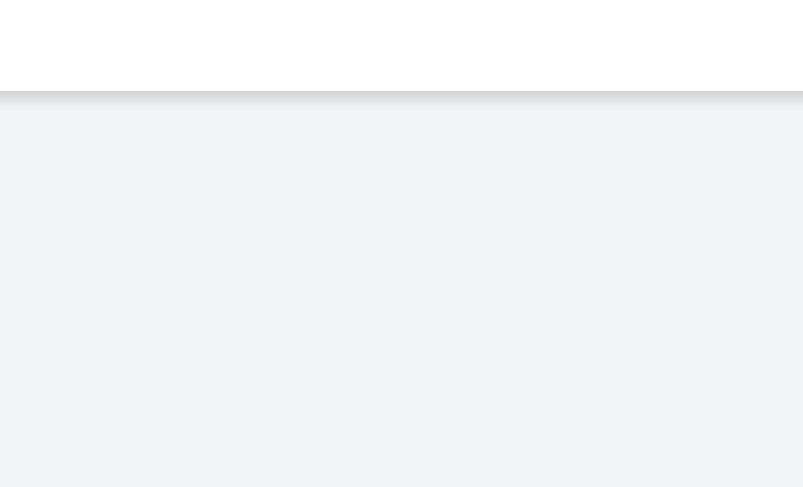
Material complementario

Los invitamos a explorar el material complementario de este curso, en esta sección encontrará recursos que le permitirán profundizar y enriquecer su aprendizaje en los temas tratados en esta unidad.

YouTube: SFPIE UV. (2023). Pensat i Dret. Inteligencia artificial y uso de los algoritmos [video]. YouTube. <https://youtu.be/utuJVI6FF5s>

YouTube: Silva Ramírez, E. (2018). Verificación formal de algoritmos: ejercicios resueltos. Servicio de Publicaciones de la Universidad de Cádiz. <https://elibro.net/es/ereader/tecnologicadeloriente/33886?page=1>

YouTube: Silva Ramírez, E. (2018). Corrección de algoritmos complejos: verificación formal. Servicio de Publicaciones de la Universidad de Cádiz. <https://elibro.net/es/ereader/tecnologicadeloriente/33887?page=1>





UNIDAD 3. DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS

SÍNTESIS

La Unidad 3: Diseño e implementación de algoritmos, se centra en el desarrollo de competencias clave para crear y optimizar algoritmos eficientes. Aprenderá a reconocer los principios de las soluciones utilizando paradigmas como divide y vencerás, programación dinámica y algoritmos voraces; implementando sus diseños en Python y aplicando técnicas de optimización asistidas por IA. Esta unidad es esencial para comprender cómo abordar problemas complejos de manera sistemática, optimizando el uso de recursos y mejorando el rendimiento de las soluciones algorítmicas en diversos contextos computacionales.

