



ChatGPT: Tu Entrenador de Python

Aprende los Fundamentos en 100 Prompts

Sergio Rojas-Galeano

ChatGPT: Tu Entrenador de Python

Aprende los Fundamentos en 100 *Prompts*

Sergio Rojas-Galeano, PhD.

Bogotá, Colombia

Sobre el Autor

Sergio Rojas-Galeano es un científico de la computación entusiasta de la Inteligencia Artificial, el Aprendizaje Automático, la Computación Evolutiva, el Procesamiento del Lenguaje Natural, la Ingeniería de Software y la Edición Científica. Con experiencia profesional tanto en posiciones académicas como industriales en Colombia e Inglaterra, está comprometido con la investigación, el avance y la promoción de los estándares éticos en la disciplina de la ciencia de la computación. Actualmente se desempeña como profesor titular en la Universidad Distrital Francisco José de Caldas (Colombia), donde ha enseñado programación e inteligencia computacional durante más de dos décadas. Con un doctorado en Ciencias de la Computación de University College London (UCL) en el Reino Unido (obtenido en 2009), también ha sido investigador visitante en UCL (en 2022) y en la Universidad de Málaga en España (en 2023). Es autor o coautor de más de 70 artículos científicos y anteriormente publicó el libro *Models of Learning and Optimisation for Data Scientists* (LeanPub, 2019) y la versión en inglés de este libro, *ChatGPT: Your Python Tutor* (LeanPub, 2023).

email: sargaleano@gmail.com

Licencia

Este documento está licenciado bajo:



El permiso para utilizar este documento está sujeto a la Licencia; cualquier otro uso está estrictamente prohibido. El contenido de este documento se proporciona "TAL CUAL", sin garantías de ningún tipo, ya sean expresas o implícitas. El autor no asume ninguna responsabilidad o responsabilidad por las consecuencias derivadas del uso de este documento, incluidos, entre otros, errores, omisiones o inexactitudes. El lector asume la responsabilidad total de la aplicación de los conceptos y técnicas descritos en este libro.

© 2023 por Sergio Rojas-Galeano.

Colofón

Este documento fue maquetado utilizando \LaTeX y la clase *kaobook*. El autor elaboró el contenido con la ayuda del sistema de generación de contenido ChatGPT de OpenAI, basado en sus propias ideas, propósito y estilo. El diseño de la portada fue creado por el autor utilizando plantillas y arte disponibles en la plataforma de diseño gráfico Canva. Todos los nombres de productos, logotipos, marcas, marcas comerciales y marcas registradas son propiedad de sus respectivos propietarios. Todos los nombres de empresas, productos y servicios, así como los nombres de celebridades, libros, personajes, bandas, clubes deportivos o películas mencionados en este libro, se utilizan únicamente con fines didácticos y de reconocimiento. El uso de estos nombres o marcas no implica respaldo alguno.

Primera edición – v1.0

Julio de 2023

Bogotá, Colombia

El código (de programación) es como un chiste.
Cuando tienes que explicarlo, es porque es malo.

– Cory House

Prefacio

¡Bienvenido a *ChatGPT: Tu Entrenador de Python. Aprende los Fundamentos en 100 Prompts!* Si estás leyendo esto, significa que estás a punto de embarcarte en un emocionante viaje al mundo de la programación. Pero, ¿por qué Python, podrías preguntar? Bueno, Python es uno de los lenguajes de programación más ampliamente utilizados y versátiles, con aplicaciones que van desde el desarrollo web y análisis de datos hasta el aprendizaje automático e inteligencia artificial (IA).

De hecho, en los últimos años ha habido un creciente interés en la IA y sus aplicaciones, incluido el desarrollo de modelos de lenguaje como ChatGPT. Si bien estos modelos admiten cuestionamientos válidos acerca de las implicaciones éticas, los sesgos de datos y la posibilidad de generar contenido falso, vale la pena señalar que en el ámbito de la programación, el contenido generado por ChatGPT consiste principalmente de código que puede evaluarse fácilmente en cuanto a su validez y efectividad simplemente ejecutándolo, con lo cual tales riesgos se ven mitigados en cierta medida. Esto ofrece una ventaja al aminorar los fallos asociados con alucinaciones o información inexacta, ya que la salida se puede verificar a través de la misma máquina, cuando la interpreta. Es así que los modelos de lenguaje como ChatGPT se han convertido en herramientas valiosas para tareas de programación, ofreciendo a los desarrolladores una asistencia provechosa para sus proyectos de codificación.

Esta guía ha sido cuidadosamente diseñada para ayudarte a navegar por los conceptos básicos de la programación en Python con facilidad, usando ChatGPT como tu mentor de confianza. ChatGPT es un potente modelo de lenguaje diseñado para ayudarte con cualquier pregunta o duda que puedas tener mientras aprendes Python. Te proporcionará ejemplos, explicaciones y retroalimentación, haciendo que tu experiencia de aprendizaje sea interactiva y entretenida.

Con ella aprenderás los fundamentos de la programación en Python, incluyendo variables, tipos de datos, operadores, estructuras de control de flujo como condicionales y bucles, contenedores como listas y diccionarios, y conceptos de modularización y reutilización. También enfrentarás desafíos divertidos y emocionantes que pondrán a prueba tus nuevas habilidades.

Pero esta guía es más que una simple colección de ejemplos y ejercicios. Es una invitación a explorar el poder de la programación y su impacto potencial en nuestras vidas, y a comprobar cómo la IA y los modelos de lenguaje como ChatGPT pueden potenciar las habilidades y productividad de los programadores para crear aplicaciones poderosas para un futuro prometedor. A medida que te adentres en el mundo de Python, verás cómo la programación se puede utilizar para resolver problemas prácticos, automatizar tareas tediosas e innovar productos y servicios.

La mayoría del contenido de este libro, incluyendo este prólogo, fue generado inicialmente con la ayuda de ChatGPT. Sin embargo, es importante señalar que el material inicial sirvió como un borrador que luego revisé, ajusté y enriquecí ampliamente. Este proceso fue mediado por mi experiencia educativa, con el objetivo de crear un contenido didáctico e informativo para los lectores. Si bien ChatGPT desempeñó un papel en la generación del material base, el contenido final ha sido cuidadosamente seleccionado y moldeado para garantizar su valor educativo y correctitud.

Para promover la transparencia en este enfoque innovador de escritura académica asistida por IA, he decidido deliberadamente incluir una lista exhaustiva de las 100 indicaciones (o *prompts* como se conocen en Inglés) que concebí y utilicé para alimentar a ChatGPT, asegurando una cobertura completa del alcance temático previsto en esta guía. Esta lista se incorporará al final del libro para una fácil referencia. Además, esta *Lista de Prompts* habilita a los lectores para que puedan eventualmente reproducir una versión similar y personalizada del libro, que se ajuste a sus preferencias y necesidades individuales.

Además, para referir a cada *prompt* utilizaré citaciones numéricas correspondientes a su posición en la lista, en cada lugar del texto donde lo utilice. Por ejemplo, el primer *prompt* utilizado para generar el borrador inicial de este prólogo se cita seguidamente¹.

Por último, he creado un repositorio complementario que aloja una colección de *Jupyter Notebooks* que contienen los ejemplos de código de Python del libro, cuidadosamente organizados por capítulos para una navegación y exploración conveniente:

<https://github.com/Sargaleano/chatgpt-python-coach>

Además, a partir del Capítulo 4, he agregado una característica llamativa, para los ejemplos de código más interesantes. Se trata de enlaces al sitio web [PythonTutor.com](https://pythontutor.com). Estos enlaces te permiten visualizar la ejecución paso a paso del código, observando dinámicamente el estado de la memoria del programa. Busca estos enlaces en la parte inferior de las listas de código y los recuadros de salida. ¡Es una forma fantástica de mejorar tu comprensión y darle vida al código!

Así que, si estás ansioso por aprender a programar y comenzar a crear programas con Python y ChatGPT, espero que encuentres esta guía útil y placentera de leer.

Sergio Rojas-Galeano
Bogotá, Julio 03, 2023

Índice general

Prefacio	vii
Índice general	ix
Introducción	3
¿Es este el libro adecuado para ti?	3
Acerca de Python	3
El ciclo de programación	4
Las herramientas	5
Uso de Colab	6
Uso de PythonTutor	9
ChatGPT como entrenador de programación	10
Uso de ChatGPT	12
Plantilla de <i>prompts</i> para ChatGPT	14
1 Un Vistazo a lo Esencial	17
1.1 ¡Hola, mundo!	17
1.2 Cadenas de caracteres y formato de cadenas	19
1.3 ¡Manos a la obra!	25
2 Datos y Variables en Acción	29
2.1 Tipos de datos	29
2.2 Entrada/salida básica	32
2.3 Un compañero de programación	34
2.4 ¡Manos a la obra!	38
3 La Sala de Operaciones	41
3.1 Operadores aritméticos	41
3.2 Operadores de comparación	43
3.3 Operadores lógicos	45
3.4 Operadores a nivel de bit	47
3.5 Operadores de asignación	51
3.6 Operadores de identidad	52
3.7 Operadores de pertenencia	54
3.8 Operador de exponenciación	55
3.9 Operadores matemáticos avanzados	56
3.10 ¡Manos a la obra!	58
4 La Torre de Control	61
4.1 Sentencia condicional if/else	62

4.2	Sentencia match/case	66
4.3	Sentencia de bucle for	68
4.4	Sentencia de bucle while	72
4.5	Estructuras de control anidadas	77
4.6	¡Manos a la obra!	84
5	Zona de Contenedores	95
5.1	Listas	95
5.2	Tuplas	103
5.3	Conjuntos	108
5.4	Cadenas de caracteres	112
5.5	Diccionarios	118
5.6	¡Manos a la obra!	132
6	Soluciones Modulares	139
6.1	Modularización y reusabilidad	139
6.2	Funciones	142
6.3	Clases y objetos	150
6.4	¡Manos a la obra!	161
7	Acto Final	169
7.1	¿Qué sigue a continuación?	169
7.2	Lecturas sugeridas	175
	Lista de los 100 <i>prompts</i>	177

Índice de figuras

1	Ciclo de desarrollo de programas	4
2	Pantalla de inicio de Colab	7
3	Un cuaderno sin título	7
4	Un cuaderno con celdas de texto y código	8
5	Página web de PythonTutor	9
6	Pantalla de visualización de PythonTutor	10
7	Página de bienvenida de ChatGPT	12
8	Página de inicio de sesión de ChatGPT	12
9	Página principal de ChatGPT	13
10	Página de chat de ChatGPT	13
7.1	Ruta de aprendizaje sugerida para el desarrollo web con Python.	169
7.2	Ruta de aprendizaje sugerida para la ciencia de datos con Python.	170
7.3	Ruta de aprendizaje sugerida para la automatización y el scripting con Python.	170
7.4	Ruta de aprendizaje sugerida para el desarrollo de juegos con Python.	171
7.5	Ruta de aprendizaje sugerida para DevOps con Python.	171
7.6	Ruta de aprendizaje sugerida para conceptos avanzados de Python.	172
7.7	Ruta de aprendizaje sugerida para pruebas y depuración en Python.	172
7.8	Ruta de aprendizaje sugerida para recursos adicionales sobre Python.	173

Introducción

¿Es este el libro adecuado para ti?

Este libro está diseñado para ayudar a personas como tú, que se están iniciando en la programación, a aprender los conceptos básicos con facilidad, específicamente en el lenguaje Python, uno de los lenguajes de programación más versátiles y ampliamente utilizados.

Pero este libro es más que una guía de programación típica. Es como tener un entrenador dedicado a tu lado, guiándote en el viaje de aprendizaje con ejemplos, explicaciones y retroalimentación. Y ese entrenador resulta ser un modelo de lenguaje de IA: ChatGPT. Con ChatGPT como tu guía, tendrás acceso a una poderosa herramienta que te ayudará a comprender los conceptos de programación esenciales. Recibirás retroalimentación en tiempo real que te ayudará a aprender más rápido y de manera más efectiva, lo que en última instancia te permitirá convertirte en un programador competente.²

Este libro sirve como una guía introductoria a conceptos fundamentales en Python, incluyendo algoritmos, estructuras de control de flujo, contenedores y modularización. Dominar estos fundamentos te proporcionará una base sólida para abordar temas más avanzados en futuros volúmenes. Así que, asimila estos conceptos fundamentales y prepárate para los emocionantes desafíos que te esperan. Mientras tanto, este capítulo presenta las herramientas que se utilizarán a lo largo del libro. Si ya estás familiarizado/a con estas herramientas y deseas comenzar a programar directamente, puedes omitir este capítulo y pasar al Capítulo 1.

Acerca de Python

Python es un lenguaje de programación versátil y amigable para principiantes. Fue creado a finales de la década de 1980 por un programador holandés llamado Guido van Rossum, quien lo nombró en honor al grupo de comedia británico, Monty Python.³

Hoy en día, Python es utilizado para el desarrollo de software en numerosas organizaciones, desde la NASA hasta Instagram, para equipar desde rovers espaciales hasta filtros fotográficos. De hecho, es uno de los lenguajes de programación más populares en el mundo, y por una buena

Contenido de Capítulo

¿Es este libro adecuado para ti?.....	3
Acerca de Python.....	3
El ciclo de programación.....	4
Las herramientas.....	5
Uso de Colab.....	6
Uso de PythonTutor.....	9
ChatGPT como entrenador.....	10
Uso de ChatGPT.....	12
Plantillas de prompts.....	14

Monty Python

Monty Python es un grupo de comedia británico conocido por su humor único y absurdo. Su programa de televisión, Monty Python's Flying Circus, se emitió desde 1969 hasta 1974 y ganó un culto de seguidores. También crearon varias películas populares, incluyendo Monty Python and the Holy Grail y Life of Brian. Su influencia en la comedia y la cultura popular ha sido significativa y continúa siendo celebrada por fans de todo el mundo.⁴

razón: es un lenguaje fácil de aprender pero con capacidades poderosas, lo que lo hace ideal tanto para principiantes como para expertos.

El futuro de Python es prometedor, ya que se espera que se vuelva aún más popular en los próximos años debido a su creciente uso en áreas como el aprendizaje automático, la inteligencia artificial y la ciencia de datos.

Existen numerosos recursos de Python disponibles en línea. El sitio web oficial de Python (<https://www.python.org/>) tiene una gran cantidad de información, incluyendo tutoriales, documentación y descargas. Otros sitios web útiles incluyen Real Python (<https://realpython.com/>), Python for Beginners (<https://www.pythonforbeginners.com/>), y Codecademy (<https://www.codecademy.com/learn/learn-python>).

El ciclo de programación

La programación es un proceso creativo que requiere una planificación cuidadosa, enfoque y atención al detalle. Ya sea que estés escribiendo un módulo simple o una aplicación compleja, existe un ciclo estándar de desarrollo de programas que debes seguir para asegurarte de que tu código funcione como se pretende. Los cuatro pasos clave de este ciclo se muestran en la Figura 1.⁶ Revisemos pues, cada paso en detalle:⁷

- **Escribir (codificar):** El primer paso es escribir el código. Aquí es donde conviertes tus ideas en instrucciones reales que la computadora puede entender. Utilizarás un lenguaje de programación como Python para escribir tu código, y deberás prestar mucha atención a la sintaxis y estructura del mismo.

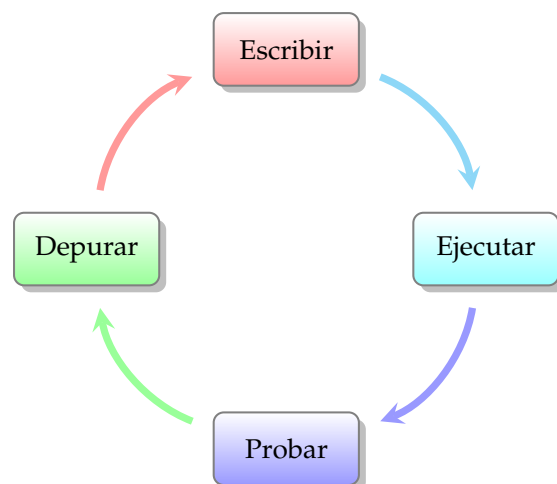


Figura 1: Un ciclo típico de desarrollo de programas

- **Ejecutar:** Una vez que hayas escrito tu código, es hora de ejecutarlo. Aquí es donde observas tu código en acción y pruebas si funciona como se pretende. Puedes utilizar un IDE (entorno de desarrollo integrado) para ejecutar tu código, o puedes utilizar una interfaz de línea de comandos para ejecutarlo.
- **Probar:** Las pruebas son una parte esencial del proceso de desarrollo de programas. Aquí es donde verificas si tu código funciona correctamente e identificas cualquier error o fallo. Deberás ejecutar diferentes casos de prueba y escenarios para asegurarte de que tu código sea robusto y funcione en una variedad de situaciones.
- **Corregir (depurar):** Por último, si encuentras errores o fallos durante las pruebas, es hora de corregirlos. Esto se conoce como depuración. Deberás volver a tu código e identificar la causa del error. Esto puede implicar recorrer tu código línea por línea o utilizar herramientas de depuración para rastrear el flujo de ejecución. Una vez que hayas identificado el problema, puedes realizar los cambios necesarios en tu código para solucionarlo.

Recuerda, la programación es un viaje de aprendizaje y mejora continua. Se requiere tiempo y esfuerzo para desarrollar buenas habilidades de programación. Acepta los desafíos que se presenten en tu camino y no temas pedir ayuda cuando sea necesario. Al adherirte a estos principios y aplicar disciplina a tu oficio, estarás en camino de crear software que no solo satisfaga las necesidades de tus usuarios, sino que también pueda tener un impacto positivo en el mundo.

Las herramientas

Python es un lenguaje de programación interpretado, lo que significa que el código se ejecuta directamente línea por línea, sin necesidad de una etapa de compilación. Esto es diferente de los lenguajes compilados como C++, Java o Go, donde el código fuente se compila en código máquina que puede ser ejecutado por una computadora.⁸

Una ventaja de un lenguaje interpretado como Python es que permite un desarrollo y prototipado más rápidos. Dado que el código puede ejecutarse de inmediato, los desarrolladores pueden probar y depurar rápidamente sus programas. Sin embargo, los lenguajes interpretados pueden ser más lentos en la ejecución que los lenguajes compilados, ya que el código debe ser reinterpretado cada vez que se ejecuta. Esto puede ser una preocupación para programas que requieren un alto rendimiento o que manejan grandes conjuntos de datos.

Versiones de Python

Algunas diferencias clave entre las versiones recientes son:⁵

- Python 2.x tiene soporte limitado para símbolos de idiomas diferentes al Inglés, mientras que Python 3.6+ utiliza codificación Unicode para soportar otros idiomas.
- Python 3.6+ permite un extenso formato de texto para imprimir resultados, en comparación con las versiones anteriores.
- Python 2.x devuelve enteros para operandos enteros, mientras que Python 3.6+ proporciona tanto la división decimal predeterminada como un comando de división entera por separado.
- Las versiones Python 3.5 y posteriores permiten administrar múltiples tareas de forma concurrente.
- Python 3.10 incorpora una nueva instrucción de chequeo de múltiples condiciones con un patrón de coincidencia.
- Python 3.11 es la versión más reciente (abril de 2023).
- Para obtener información actualizada sobre las versiones de Python, consulta: <https://devguide.python.org/versions/>.

Existen numerosos Entornos de Desarrollo Integrados (IDE) de Python disponibles para llevar a cabo el ciclo de desarrollo de la Figura 1. Estas son aplicaciones de software, como *PyCharm*, *Spyder* y *Visual Studio Code*, que ofrecen funciones avanzadas para ayudar a los desarrolladores experimentados a acelerar el proceso de desarrollo y aumentar la productividad.

Sin embargo, en este libro queremos centrarnos en los conceptos y algoritmos detrás de la programación sin las molestias de la instalación de software. Por lo tanto, recurriremos a herramientas en línea listas para usar como nuestra plataforma principal para escribir, ejecutar y probar nuestro código Python. Estas herramientas se detallan a continuación:

- **Jupyter Notebooks:** un entorno computacional interactivo basado en la web que te permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Sitio web: <https://jupyter.org/>. No se requiere una cuenta para usarlo.
- **Google Colaboratory (Colab):** un entorno gratuito proporcionado por Google para trabajar con Jupyter Notebooks en la nube. Te permite ejecutar código Python en tu navegador sin necesidad de ninguna configuración. Para ejecutar Colaboratory, simplemente debes iniciar sesión en tu cuenta de Google, ir al sitio web (<https://colab.research.google.com/>) y crear un nuevo cuaderno de trabajo. A partir de ahí, puedes comenzar a escribir código Python y ejecutarlo en tiempo real. Tu trabajo se guardará en tu cuenta de Google Drive.
- **PythonTutor:** un visualizador basado en la web para código Python que ayuda a los estudiantes a comprender cómo se ejecuta el código por el intérprete de Python. Para usar PythonTutor, debes ir al sitio web (<http://pythontutor.com/>); desde allí, puedes pegar tu código Python o escribirlo directamente en el sitio y seguidamente visualizar su ejecución. No se requiere una cuenta para usarlo.
- **ChatGPT:** un modelo de lenguaje que puede ayudar en el aprendizaje, codificación, explicación, depuración, documentación y proporcionar comentarios sobre programas Python. Sitio web: <https://chat.openai.com>. Requiere una cuenta de OpenAI.

Uso de Colab

Veamos cómo crear un nuevo cuaderno Jupyter en Google Colab, donde podemos llevar a cabo programación básica en Python.

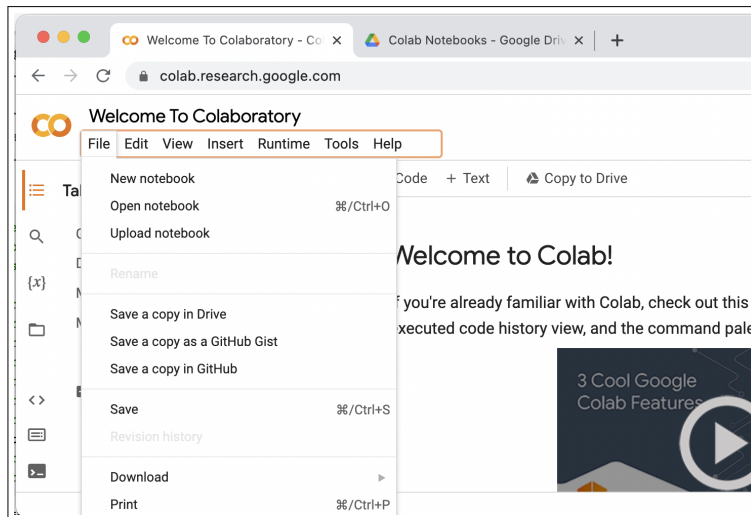


Figura 2: Pantalla de inicio de Colab.

En primer lugar, inicia sesión en tu cuenta personal de Google (en caso de que lo necesites, consulta este enlace para crear una: <https://support.google.com/accounts/answer/27441?hl=es>). Luego, sigue estos pasos:⁹

1. Ve a <https://colab.research.google.com/> en tu navegador web para abrir Colab. Aparecerá la pantalla que se muestra en Figura 2.
2. Selecciona *File* → *New notebook* en la barra de menú. Esto creará un nuevo cuaderno sin título. Puedes darle un título haciendo clic en el texto “Untitled0.ipynb” en la parte superior de la página y escribiendo un nuevo nombre, como se muestra en Figura 3.
3. Ahora puedes comenzar a escribir el contenido en las celdas del cuaderno. Hay dos tipos de celdas en Jupyter Notebook:¹⁰

- **Celdas de código:** Estas celdas permiten al usuario escribir y ejecutar código en el cuaderno. Las celdas de código se

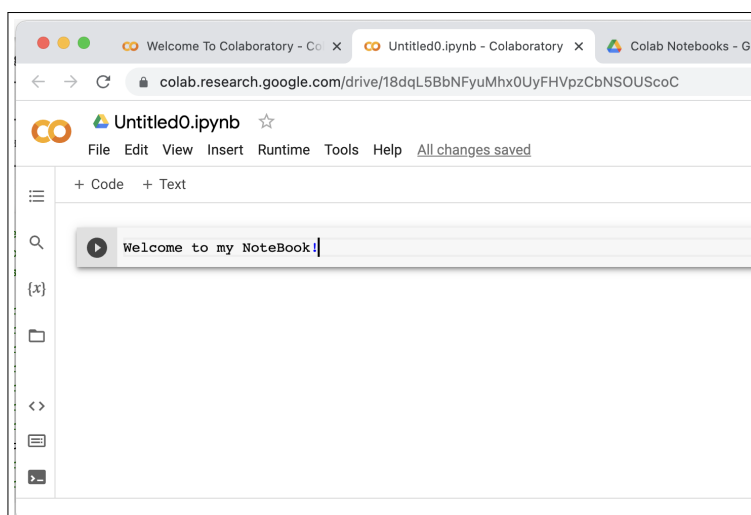


Figura 3: Un cuaderno sin título.

denotan por el indicador **In []** en el lado izquierdo de la celda. Cuando ejecutas el código en una celda, la salida se muestra en un cuadro de texto debajo de la celda y aparece un número entre corchetes a la izquierda de la celda que indica en qué orden se ejecutó. Colab admite una variedad de lenguajes de programación, pero el predeterminado es Python.

- **Celdas de Markdown:** Estas celdas permiten al usuario agregar texto, imágenes y otros tipos de contenido multimedia al cuaderno. Las celdas de Markdown utilizan el lenguaje Markdown, que es un lenguaje de marcado ligero que permite formatear el texto fácilmente utilizando una sintaxis sencilla. Cuando ejecutas una celda de Markdown, el texto se visualizará como texto formateado o contenido multimedia.
4. Puedes ejecutar el código en la celda haciendo clic en el botón ▶ en el lado izquierdo de la celda o utilizando el atajo de teclado *Shift+Enter*.
 5. Si necesitas agregar más celdas, puedes hacer clic en el botón + en la parte superior izquierda de la página o ir al menú *Insert* → *Code cell* o *Insert* → *Text cell*.
 6. Para guardar tu cuaderno, puedes hacer clic en *File* → *Save* o utilizar el atajo de teclado *Ctrl+S*.
 7. Colab guarda automáticamente tu cuaderno en tu Google Drive, por lo que puedes acceder a él desde cualquier dispositivo siempre que hayas iniciado sesión en la misma cuenta de Google.

La Figura 4 muestra un ejemplo de un cuaderno que incluye una celda de Markdown con texto introductorio y una celda de código con un programa “Hello, World!”. También se puede ver la salida después de ejecutar el código.

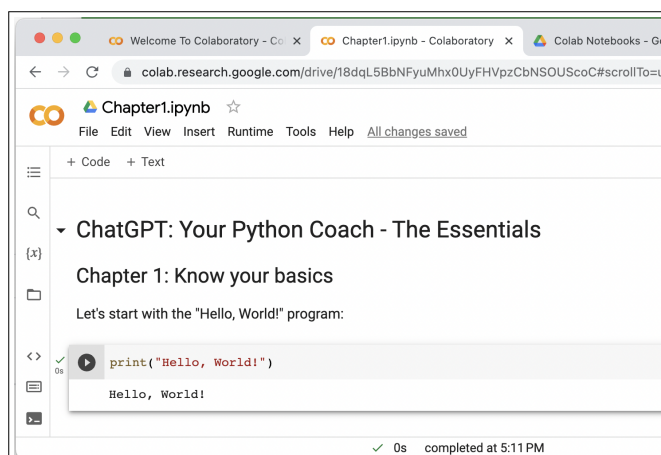


Figura 4: Un cuaderno con celdas de texto y código.

Uso de PythonTutor

PythonTutor es una herramienta en línea que te permite visualizar la ejecución del código de Python paso a paso. Puede ayudarte a comprender el funcionamiento de tu programa mostrándote cómo se ejecuta línea por línea y cómo cambian los valores de las variables a medida que se ejecuta.¹¹

Para usar PythonTutor, debes seguir estos pasos:

1. Ve al sitio web de PythonTutor en www.pythontutor.com. Aparecerá la pantalla mostrada en la Figura 5. Seguidamente, haz clic en *Start coding now in Python*.

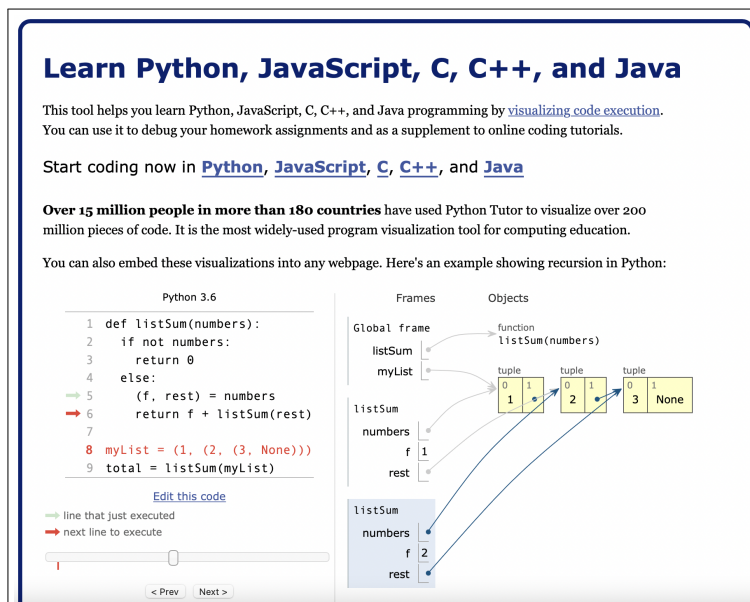


Figura 5: Página web de PythonTutor, donde puedes visualizar y seguir la ejecución del código de Python.

2. Elige el lenguaje de Python haciendo clic en el botón correspondiente. Selecciona la versión de Python 3.6.
3. Ingresa tu código de Python en el cuadro de texto proporcionado.
4. Selecciona las opciones de visualización haciendo clic en las casillas de verificación y menús desplegables proporcionados. Puedes elegir ver la salida del código, los valores de las variables y la pila de llamadas a módulos, entre otras opciones.
5. Haz clic en el botón *Visualize Execution* para iniciar la visualización.
6. Una vez que comience la visualización, verás una línea de tiempo en el lado izquierdo de la pantalla que muestra cada paso en la ejecución de tu código. Puedes avanzar en el código una línea a la vez haciendo clic en los botones *Next* o *Prev*, o usar el control deslizante para recorrer la ejecución paso a paso. Un ejemplo de la visualización de un programa se puede ver en la Figura 6.

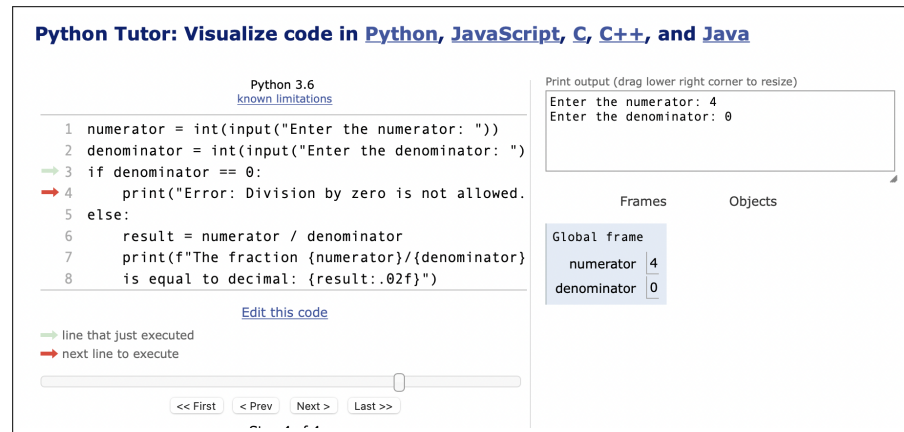


Figura 6: Pantalla de visualización de PythonTutor.

7. A medida que avanzas en el código, el lado derecho de la pantalla mostrará el estado actual del programa, incluyendo los valores de las variables y cualquier salida generada por el código.
8. También puedes interactuar con la visualización haciendo clic en el código o las variables para ver información más detallada.

ChatGPT como entrenador de programación

Declaración

Este libro fue escrito con la ayuda de ChatGPT. A lo largo del libro, fragmentos de contenido generados automáticamente se utilizaron como borradores, los cuales revisé, verifiqué, ajusté, remezclé y amplié hasta alcanzar su forma final. Con el fin de ser transparente, también he incluido las indicaciones (*prompts*) que utilicé para alimentar el modelo y generar los borradores, los cuales están marcados con un número en superíndice que se encuentra justo después del fragmento de texto final donde se usaron. Estos números indican su posición en la *Lista de Prompts* proporcionada al final del libro.

ChatGPT es un modelo de lenguaje creado por OpenAI que está diseñado para comunicarse con los humanos utilizando lenguaje natural. Es un tipo de inteligencia artificial que puede entender y responder por medio de texto de manera conversacional. Ha sido entrenado con una gran cantidad de datos para poder entender y generar lenguaje natural, incluyendo lenguajes de programación estructurados de alto nivel, lo que lo convierte en una excelente herramienta para ayudar a los estudiantes a aprender programación de una manera dialogada y entretenida.¹²

En el proceso de aprendizaje de la programación, ChatGPT puede servir como un entrenador bastante útil. Puedes hacerle preguntas sobre conceptos de programación, solicitar ejemplos de código e incluso obtener comentarios sobre tu propio código. Puede proporcionar explicaciones y aclaraciones sobre temas de programación y servir como una herramienta para practicar habilidades de codificación.

Concretamente, aquí hay algunas formas en las que ChatGPT puede ayudarte a aprender programación:

- ▶ Puede proporcionar explicaciones y ejemplos de conceptos y sintaxis de programación, ayudándote a comprenderlos mejor.
- ▶ Puede responder preguntas de codificación, como por qué se necesita una regla de sintaxis específica o cómo realizar una tarea en particular.

- ▶ Puede ayudarte a depurar tu código sugiriendo posibles soluciones y explicando por qué ocurren ciertos errores.
- ▶ Puede ayudarte a aprender a escribir código eficiente y optimizado al brindarte comentarios sobre el rendimiento de tu código.
- ▶ Puede sugerir recursos y materiales de aprendizaje para ayudarte a ampliar tu comprensión de la programación.
- ▶ Puede ayudarte a documentar tu código para que sea más fácil de entender para otros programadores o para que puedas recordarlo fácilmente en futuras ocasiones.
- ▶ Puede traducir código entre diferentes lenguajes de programación para crear nuevas versiones de tus programas.
- ▶ Puede proporcionar referencias contextuales sobre el significado, la historia o las anécdotas relacionadas con el desarrollo de la programación y la informática en general.

Aunque ChatGPT puede ser una herramienta valiosa para aprender programación, no se debe depender exclusivamente de ella como única fuente de información o retroalimentación. Es importante utilizarla de manera responsable, verificar las respuestas con otras fuentes y ser consciente de posibles inexactitudes o sesgos en los datos utilizados para entrenar este modelo de lenguaje de IA.

Respecto a los cuestionamientos éticos relacionados con el uso de ChatGPT, es fundamental tener en cuenta que no puede reemplazar habilidades clave que necesitas adquirir como programador talentoso, especialmente la resolución de problemas y el pensamiento crítico. Si bien ChatGPT puede ser un excelente compañero en tu viaje de aprendizaje o incluso un compañero de codificación cuando te embarques en tus proyectos de programación, no puede reemplazar (al menos no al momento de escribir este libro) los beneficios que puedes obtener de la ingeniosidad, la interacción y la orientación humana.

Además, pronto te darás cuenta de la importancia de perfeccionar tu habilidad para formular preguntas bien estructuradas y aprovechar al máximo ChatGPT en esta experiencia de *coaching*. Esto es lo que se ha venido a conocer como *prompt engineering*, que consiste en indicar claramente al modelo en qué necesitas ayuda, de manera que se reduzcan las inexactitudes o resultados falsos.

En general, ChatGPT es un recurso increíble para mejorar tus habilidades de programación, pero úsalo de manera responsable. Con una mentalidad crítica, podrás aprovechar su potencial para convertirte en un programador competente y comenzar una carrera prometedora en este campo.

Uso de ChatGPT

A continuación se presenta una breve descripción de cómo usar ChatGPT.

1. En tu navegador web visita chat.openai.com. Una vez que se cargue la página, deberías ver la pantalla mostrada en Figura 7.

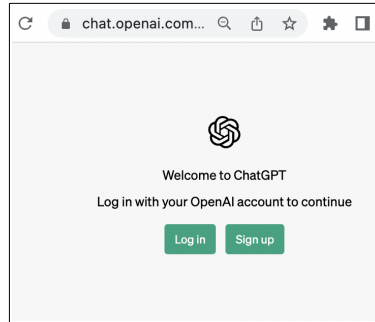


Figura 7: Página de bienvenida de ChatGPT.

2. Inicia sesión en tu cuenta de OpenAI utilizando la pantalla que se muestra en Figura 8. Si no tienes una cuenta, elige la opción *Sign up* y sigue las instrucciones.

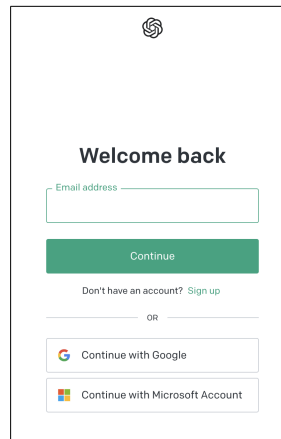


Figura 8: Página de inicio de sesión de ChatGPT.

3. Una vez que hayas iniciado sesión, serás llevado a la página principal de ChatGPT, que se muestra en Figura 9. En el cuadro de texto *Send a message.*, ingresa la pregunta (es decir, el *prompt*) que deseas enviar a ChatGPT y luego presiona la tecla *Enter* o haz clic en el botón de flecha.
4. Espera a que ChatGPT genere una respuesta. La respuesta aparecerá en la ventana de chat debajo de tu mensaje. Ten en cuenta que la respuesta puede incluir texto explicativo, así como código de programación. Puedes hacer clic en el botón *Copy code*, como se ilustra en la captura de pantalla en Figura 10, para copiar el código y pegarlo en una plataforma como Colab o PythonTutor para ejecutar el programa.

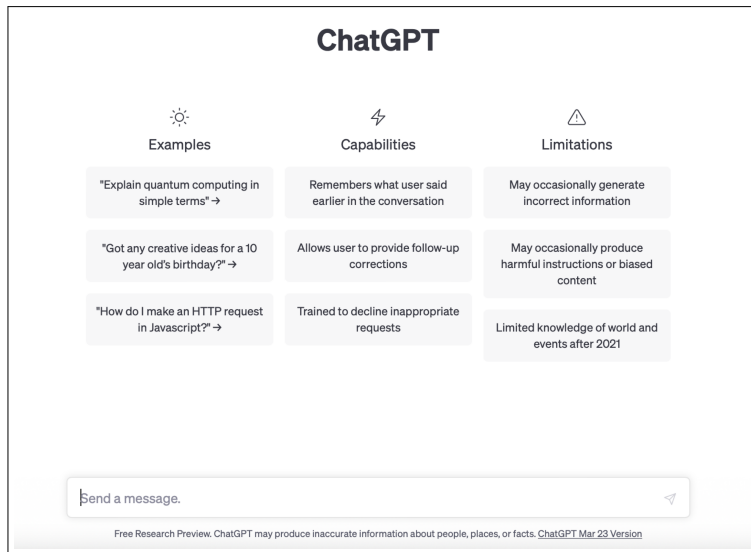


Figura 9: Página principal de ChatGPT.

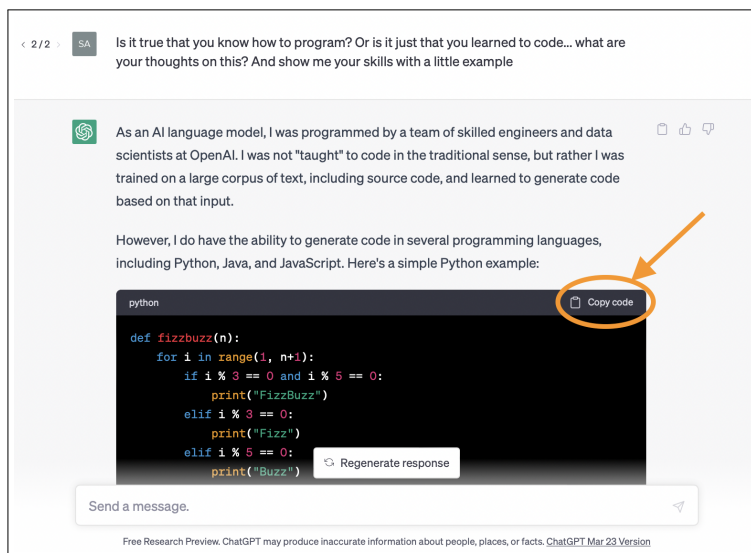


Figura 10: Página de chat de ChatGPT.

5. Si la respuesta no responde completamente tu pregunta o necesitas más información, escribe otro *prompt*, envíalo a ChatGPT y continúa la conversación. El historial de conversaciones anteriores se mantiene en el lado izquierdo de la pantalla, por lo que puedes regresar y retomar cualquiera de ellas si lo deseas.

En este libro, utilizaré ChatGPT como se describió anteriormente. Para hacer el material más legible, formatearé los *prompts*, texto explicativo, código y salida de la respuesta usando diferentes cajas de texto de colores. A modo de ilustración, he reproducido la captura de pantalla anterior en el estilo de formato mencionado (traducido al Español):

¿Es cierto que sabes programar? ¿O es solo que aprendiste a codificar... qué piensas al respecto? Y muéstrame tus habilidades con un pequeño ejemplo.

Como modelo de lenguaje de IA, fui programado por un equipo de ingenieros y científicos de datos expertos en OpenAI. No fui “enseñado” a codificar en el sentido tradicional, sino que fui entrenado con un gran corpus de texto, incluido código fuente, y aprendí a generar código basado en esa entrada.

Aquí tienes un ejemplo de código con su salida:

```
1 def fizzbuzz(n):  
2     for i in range(1, n+1):  
3         if i % 3 == 0 and i % 5 == 0:  
4             print("FizzBuzz")  
5     ...
```

```
1  
2  
Fizz  
4  
Buzz  
...
```

Plantillas de *prompts* para el entrenador ChatGPT

Como se mencionó anteriormente, crear preguntas efectivas al solicitar ayuda a ChatGPT para la programación es una habilidad que se desarrollará a medida que interactúes con la herramienta y alcances metas intermedias. En los siguientes capítulos, utilizaremos las siguientes plantillas de preguntas para cubrir diversos temas de programación:¹³

1. Presentar y explicar conceptos de programación:
 - ▶ “Explique [concepto de programación] de manera sencilla.”
 - ▶ “¿Cuál es la intuición detrás del [concepto de programación]?”
 - ▶ “Proporcione una analogía de la vida real del [concepto de programación].”
2. Escribir ejemplos de código relacionados con conceptos de programación:
 - ▶ “Escriba un ejemplo de código para [concepto de programación].”
 - ▶ “¿Cuál es una aplicación práctica del [concepto de programación]?”
 - ▶ “¿Cómo se utiliza [concepto de programación] para resolver [algún problema]?”
 - ▶ “Ilustre el uso de [varios conceptos de programación] en un solo ejemplo de código, que [utilice/calcula/compara] [datos/información/estadísticas] sobre [tema de la vida real]. Por favor, incluya comentarios explicativos en cada bloque de código.”

3. Describir las líneas clave del código:

- ▶ “¿Qué significa esta línea de código: [fragmento de código]?”
- ▶ “¿Cómo contribuye [línea específica de código] a la funcionalidad del programa?”
- ▶ “¿Qué efecto tendría cambiar [línea específica de código] por [código diferente]?”

4. Documentar y comentar código:

- ▶ “Escriba cadenas de documentación (*docstrings*) para mi código de Python.”
- ▶ “Agregue comentarios en línea para describir la funcionalidad de bloques de código en mi programa.”

5. Generar nuevos desafíos:

- ▶ “Sugiera algunos ejercicios similares a [problema].”
- ▶ “¿Qué variaciones de [problema] puedo intentar resolver?”
- ▶ “¿Cómo puedo practicar [concepto de programación] en diferentes escenarios?”

6. Incluir enlaces útiles a recursos:

- ▶ “¿Recomienda algunos recursos para aprender [concepto de programación]?”
- ▶ “¿Dónde puedo encontrar más información sobre [concepto de programación]?”
- ▶ “¿Cuáles son algunos tutoriales en línea útiles sobre [concepto de programación]?”

7. Hacer el tema más agradable:

- ▶ “Cuénteme una anécdota de programación relacionada con [concepto de programación].”
- ▶ “Enumere algunos datos curiosos sobre [concepto de programación].”

Entonces, sin más dudas, sumerjámonos directamente en nuestra emocionante aventura de aprender Python con la ayuda de ChatGPT. Espero que encuentres esta experiencia divertida y gratificante.

Un Vistazo a lo Esencial

1

1.1 ¡Hola, mundo!

Explica el concepto del programa “¡Hola, mundo!” de una manera sencilla pero colorida, sin utilizar código. Además, incluye una anécdota relacionada para proporcionar a los principiantes en programación una introducción cálida al tema.¹⁴

¡Ah, el legendario programa *¡Hola, mundo!*! Esta es la forma perfecta de comenzar tu viaje en la programación, y es una experiencia que siempre recordarás. Imagínate sentado frente a tu computadora, sintiendo una mezcla de emoción y nerviosismo. ¡Estás a punto de escribir tu primer programa en Python!

El programa *¡Hola, mundo!* es el programa más sencillo que puedes escribir, pero es un evento significativo. Es el equivalente al primer llanto de un recién nacido o a la primera nota de un músico en un nuevo instrumento. Este programa es la base de todos los lenguajes de programación, y es el punto de partida perfecto para cualquiera que quiera aprender a programar.

Ahora, podrías preguntarte, “¿Por qué cominos querría escribir un programa que simplemente diga: *¡Hola, mundo!*?” Bueno, no se trata del mensaje en sí, sino del proceso de escribir el programa. Este programa sencillo es una excelente manera de familiarizarte con un nuevo lenguaje, superar el miedo a comenzar algo nuevo y ganar la confianza para seguir aprendiendo y explorando.

*Pero, ¿por qué el mensaje *¡Hola, mundo!*? ¿Quién lo inventó? ¿Cuál es su significado?¹⁵*

El origen del programa *¡Hola, mundo!* es algo incierto, pero se cree que se originó en los Bell Labs a principios de la década de 1970. La primera instancia registrada de la frase utilizada en un contexto de programación fue en un libro de 1974 sobre el lenguaje de programación C escrito por Brian Kernighan y Dennis Ritchie.

En cuanto al significado detrás del mensaje, simplemente es una forma para que los programadores prueben que su programa está funcionando correctamente y puede mostrar una salida. Es una tradición y un rito de iniciación para los principiantes en el mundo de la programación.

Bueno, suficiente de palabras. Vamos a sumergirnos en el código y comenzar la experiencia de entrenamiento con ChatGPT.

¡En sus marcas, listos... código!

Escribe el clásico programa '¡Hola, mundo!' en Python y explícalo. Proporciona también la salida. Utilizaré Colab.¹⁶

Aquí está el clásico programa ¡Hola, mundo! en Python:

```
1 print("¡Hola, mundo!")
```

La función `print()` es una función incorporada en Python que te permite mostrar texto u otros datos en la consola, en este caso el texto (también conocido como *cadena*) "¡Hola, mundo!". La salida debería verse así:

```
iHola, mundo!
```

Para ejecutar este programa en Google Colab, sigue estos pasos:

1. Haz clic en el botón *Copy code* en ChatGPT.
2. Pégallo en la primera celda de tu cuaderno en Colab.
3. Haz clic en el botón ▷ junto a la celda o presiona *Shift + Enter*.

Explica qué es un error de sintaxis y muestra 3 posibles errores de sintaxis en el programa '¡Hola, mundo!' con sus respectivas salidas.¹⁷

Cuando escribes un programa, debes seguir las reglas de sintaxis correcta del lenguaje de programación. Un error de sintaxis ocurre cuando el código que escribes no cumple con dichas reglas, como omitir un signo de puntuación, utilizar mayúsculas incorrectas para una palabra clave o escribir un operador incorrecto. Estos errores evitan que el código se interprete correctamente. A continuación tres posibles errores de sintaxis que podrían haber ocurrido en el programa ¡Hola, mundo!:

1. Paréntesis faltante:

```
print("iHola, mundo!"
File "<stdin>", line 1
print("iHola, mundo!"
                        ^
SyntaxError: unexpected EOF while parsing
```

El mensaje de error nos dice que se encontró el final inesperado del archivo (EOF) mientras se analizaba el código.

2. Palabra clave mal escrita:

```
prnt("iHola, mundo!")
File "<stdin>", line 1
  prnt("iHola, mundo!")
    ^
NameError: name 'prnt' is not defined
```

Aquí, el mensaje de error señala que no existe el comando `prnt()`.

3. Comillas faltantes:

```
print(iHola, mundo!)
File "<stdin>", line 1
print(iHola, mundo!)
          ^
SyntaxError: invalid syntax
```

Aquí, el error indica que el mensaje de la cadena debe estar entre comillas.

*¿Por qué se llama `print()`? ¿Necesito una impresora para poder usarlo?*¹⁸

La función `print()` en Python se utiliza para mostrar o imprimir un mensaje u otra salida en la consola. Se llama “print” porque originalmente se refería a imprimir información en una impresora física. Esto era en la época en que las computadoras no tenían pantallas. Sin embargo, en la actualidad, simplemente significa mostrar información en la pantalla de una computadora, y no es necesario tener una impresora para usarla.

1.2 Cadenas de caracteres y formato de cadenas

*Explica de manera sencilla qué es una cadena de caracteres y las reglas de sintaxis para definir cadenas en Python.*¹⁹

En Python, una cadena de caracteres es una secuencia de caracteres encerrados entre comillas (comillas simples o comillas dobles). Puede contener letras, números, símbolos y espacios en blanco. Las cadenas de caracteres se utilizan comúnmente para almacenar y manipular datos de texto en programación. También son conocidos como *strings*.

Para definir una cadena de caracteres en Python, simplemente encierra la secuencia deseada de caracteres entre comillas. Por ejemplo:

```
1 mi_cadena = "¡Hola, mundo!"
2 print(mi_cadena)
```

```
¡Hola, mundo!
```

En Python, puedes usar comillas simples (') o comillas dobles (") para definir una cadena de caracteres. Lo importante es ser consistente con tu elección de comillas a lo largo de tu código.

A continuación tienes algunos ejemplos de cómo definir cadenas de caracteres en Python. Cada cadena se asigna, es decir, se almacena en memoria en una variable, mediante el operador de asignación '=' y luego se imprimen.

Observa que puedes imprimir múltiples cadenas utilizando una lista de variables separada por comas. Utiliza el modificador `sep='\\n'` para imprimir en diferentes líneas de salida:

```
1 cadena1 = "Este es un string utilizando comillas dobles"
2 cadena2 = 'Este es un string utilizando comillas simples'
3 cadena3 = "Este string contiene números y símbolos: $5.99"
4 print(cadena1, cadena2, cadena3, sep='\\n')
```

```
Este es un string utilizando comillas dobles
Este es un string utilizando comillas simples
Este string contiene números y símbolos: $5.99
```

También puedes usar comillas triples (ya sean simples o dobles) para definir una cadena de múltiples líneas:

```
1 cadena_multilinea = """Esta es una
2 cadena de varias
3 líneas"""
4 print(cadena_multilinea)
```

```
Esta es una
cadena de varias
líneas
```

Además, puedes usar caracteres de escape (por ejemplo, `'\\n'` para un salto de línea, `'\\t'` para una tabulación) para incluir caracteres especiales dentro de una cadena. Por ejemplo:

```
1 cadena_escape = "Esta cadena contiene un salto de línea.\\nAquí está la siguiente línea.\\n
2 \\tY aquí hay otra línea con sangría hacia la derecha"
3 print(cadena_escape)
```

```
Esta cadena contiene un salto de línea.
Aquí está la siguiente línea.
Y aquí hay otra línea con sangría hacia la derecha
```

Explica el formato de cadenas de caracteres de manera sencilla. Escriba ejemplos de código divertidos para cada una de las opciones de formato.²⁰

El formato de cadenas de caracteres es el proceso de crear una nueva cadena de caracteres incrustando los valores de variables o expresiones en espacios reservados dentro de una cadena. Esto te permite crear cadenas dinámicas que pueden mostrar diferentes valores según la entrada que reciba.

En Python, hay varias formas de formatear cadenas:

- **f-strings**: Esta es la forma más nueva y sencilla de formatear cadenas en Python. Simplemente incluyes la variable o expresión entre llaves dentro de una cadena, precedida por la letra 'f'.
- **str.format()**: Este es un método ligeramente más antiguo de formateo de cadenas que todavía se usa ampliamente. Implica llamar al método `format()` en una cadena y pasar los valores que deseas insertar en los espacios reservados utilizando argumentos posicionales o de palabra clave.
- **operador %**: Este es un método más antiguo de formateo de cadenas que se usa menos en la actualidad. Implica usar el operador '%' para sustituir los valores de las variables o expresiones en los espacios reservados de la cadena.
- **concatenación**: En este enfoque, se utiliza el operador + para unir una sola cadena.

Aquí tienes algunos ejemplos de cada tipo de formateo:

```

1 # f-strings
2 nombre = "Alicia"
3 edad = 30
4 print(f"{nombre} tiene {edad} años y le encanta Python.")
5
6 # str.format()
7 comida = "pizza"
8 precio = 10.99
9 print("Me encanta la {}, cuesta ${:.2f} por rebanada.".format(comida, precio))
10
11 # operador %
12 ciudad = "París"
13 pais = "Francia"
14 print("Me gustaría visitar %s en %s algún día." % (ciudad, pais))
15
16 # concatenación
17 print('¡' + 'Hola, ' + 'Mundo' + '!')
```

Alicia tiene 30 años y le encanta Python!
 Me encanta la pizza, cuesta \$10.99 por rebanada.
 Me gustaría visitar París en Francia algún día.
 ¡Hola, Mundo!

¿Por qué hay algunas líneas verdes en el código que comienzan con un símbolo de numeral?²¹

Las líneas verdes en el código que comienzan con un símbolo de numeral se llaman comentarios. Los comentarios se utilizan para proporcionar notas explicativas o recordatorios

para la persona que lee el código. El intérprete de Python ignora los comentarios, por lo que no tienen ningún efecto en la ejecución del programa. Los comentarios son una forma útil para que los programadores se comuniquen entre sí, y también pueden ayudar a que el código sea más fácil de entender y mantener.

Ok. Ahora dame un ejemplo de un programa que use solo comandos print() con una analogía del mundo real.²²

Aquí tienes un ejemplo de un programa que imprime una receta en la pantalla:

```

1 # Ejemplo de programa: "Cocinando Panqueques"
2 # Este programa muestra los pasos para cocinar panqueques
3 print("-"*30 + " Cocinando Panqueques " + "-"*30)
4 print("Paso 1: Mezcla la masa para panqueques.")
5 print("Paso 2: Calienta la plancha o sartén a fuego medio.")
6 print("Paso 3: Vierte la masa en la plancha o sartén, usando una taza medidora de 1/4 de
   taza.")
7 print("Paso 4: Cocina durante 1-2 minutos, hasta que los bordes comiencen a secarse y la
   superficie esté burbujeante.")
8 print("Paso 5: Da vuelta el panqueque con una espátula
9
10 y cocínalo durante 1-2 minutos adicionales por el otro lado.")
11 print("Paso 6: Retira el panqueque de la plancha o sartén y colócalo en un plato.")
12 print("Paso 7: Repite los pasos 3-6 para la masa restante.")
13 print("Paso 8: Sirve con tus coberturas favoritas, como mantequilla y jarabe.")

```

La salida se muestra a continuación:

```

----- Cocinando Panqueques -----
Paso 1: Mezcla la masa para panqueques.
Paso 2: Calienta la plancha o sartén a fuego medio.
Paso 3: Vierte la masa en la plancha o sartén, usando una taza medidora de 1/4 de
taza.
Paso 4: Cocina durante 1-2 minutos, hasta que los bordes comiencen a secarse y la
superficie esté burbujeante.
Paso 5: Da vuelta el panqueque con una espátula y cocínalo durante 1-2 minutos
adicionales por el otro lado.
Paso 6: Retira el panqueque de la plancha o sartén y colócalo en un plato.
Paso 7: Repite los pasos 3-6 para la masa restante.
Paso 8: Sirve con tus coberturas favoritas, como mantequilla y jarabe.

```

¿Puedo utilizar opciones de formato como ancho y alineación para construir cadenas? Muéstrame ejemplos interesantes.²³

Puedes utilizar opciones de formato de ancho y alineación para construir cadenas en Python. El ancho especifica el ancho total de la cadena, mientras que la alineación especifica cómo alinear la cadena dentro de ese ancho. Aquí tienes algunos ejemplos:

1. **Texto centrado:** Puedes centrar una cadena utilizando el patrón `{:^ancho}` para indicar la alineación central y especificar el ancho. Observa cómo se usa la expresión `'-*8` para imprimir repetidamente el carácter `'-'`.

```
1 texto = "¡Hola, Mundo!"
2 texto_centrado = f"|{texto:^40}|"
3 print('-'*8, texto_centrado, '-'*8)
4
```

```
----- |                ¡Hola, Mundo!                | -----
```

2. **Texto alineado a la izquierda:** Puedes alinear a la izquierda una cadena utilizando el patrón `{:<ancho}` para indicar la alineación a la izquierda y especificar el ancho.

```
1 nombre = "Juan"
2 edad = 30
3 trabajo = "Ingeniero"
4 resultado = f"|Nombre: {nombre:<10}|Edad: {edad:<5}|Trabajo: {trabajo:<10}|"
5 print(resultado)
6
```

```
|Nombre: Juan      |Edad: 30  |Trabajo: Ingeniero|
```

3. **Texto alineado a la derecha:** Puedes alinear a la derecha una cadena utilizando el patrón `{:>ancho}` para indicar la alineación a la derecha y especificar el ancho.

```
1 nombre = "Sara"
2 edad = 28
3 trabajo = "Diseñadora"
4 resultado = f"|Nombre: {nombre:>10}|Edad: {edad:>5}|Trabajo: {trabajo:>10}|"
5 print(resultado)
6
```

```
|Nombre:          Sara|Edad:   28|Trabajo: Diseñadora|
```

4. **Texto truncado:** Puedes truncar una cadena especificando el ancho máximo, utilizando el patrón `{:.ancho}` dentro del espacio reservado.

```
1 texto = "Este es un string muy largo que necesita ser truncado"
2 texto_truncado = f"{texto:.27}"
3 print(texto_truncado+"...")
4
```

Este es un string muy largo ...

5. **Números con relleno de ceros:** Puedes rellenar con ceros un número utilizando el patrón `{:0ancho}` para indicar el relleno con ceros y especificar el ancho.

```
1 numero = 42
2 numero_relleno = f"{numero:05}"
3 print(numero_relleno)
4
```

00042

6. **Relleno y decimales:** Para dar formato a un número de punto flotante con un número específico de decimales y relleno, se utiliza el patrón `{:ancho.precisionf}`.

```
1 num = 3.14159
2 num_formateado = "{:08.2f}".format(num)
3 print("El número formateado es:", num_formateado)
4
```

El número formateado es: 00003.14

¿Qué se entiende por número “de punto flotante”?²⁴

Un número de punto flotante es un valor numérico que tiene un punto decimal. Se utiliza para representar números reales en los lenguajes de programación. Los números de punto flotante pueden ser positivos o negativos, y pueden tener diferentes grados de precisión, según la cantidad de dígitos utilizados para representarlos. Se llaman “de punto flotante” porque el punto decimal puede “flotar” o moverse a diferentes posiciones dentro del número, según la magnitud del número. Se almacenan en la memoria de una computadora utilizando un formato estándar que permite una computación y almacenamiento eficientes. En Python, los números de punto flotante se representan utilizando el tipo de datos `float`.

1.3 ¡Manos a la obra!

Me gustaría poner en práctica lo que acabo de aprender sobre print() y el formateo de cadenas. Por favor, proporciona cuatro ejercicios de código en los que deba corregir los errores para que se ejecuten correctamente.²⁵

Aquí tienes cuatro ejercicios en los que debes corregir los errores para que se ejecuten correctamente:

Ejercicio 1:

```
1 print(¡Hola, Mundo!)
```

Ejercicio 2:

```
1 mensaje = "Tengo {} manzanas"
2 print(mensaje+format(5))
```

Ejercicio 3:

```
1 nombre = input("¿Cuál es tu nombre? ")
2 edad = input("¿Cuántos años tienes? ")
3 print("Tu nombre es {} y tienes {} años."format(nombre, edad))
```

Ejercicio 4:

```
1 numero = 42
2 print("La respuesta es {:.10}".format(numero))
```

Me gustaría poner en práctica lo que acabo de aprender sobre print() y el formateo de cadenas. Esta vez, sugiere algunos ejercicios que se puedan resolver utilizando solo print() y el formateo de cadenas.²⁶

Ejercicio 1:

Imprime un recibo de compra como el que se muestra a continuación.

```
-----
|          RECIBO          |
| Artículo    | Cantidad | Precio|
|-----|-----|-----|
| Manzanas    | 3        | $1.50 |
| Bananas     | 2        | $0.80 |
| Naranjas    | 4        | $2.00 |
|-----|-----|-----|
```

Ejercicio 2:

Crea un programa que imprima un triángulo rectángulo utilizando el carácter '*' como se muestra a continuación.

```
*
* *
* * *
* * * *
* * * * *
```

Ejercicio 3:

Crea un programa que imprima la siguiente suma en la pantalla:

```
  1
 20
300
+ 4000
-----
4321
```

Un aspecto interesante de ChatGPT es su capacidad para revisar, corregir y proporcionar comentarios sobre un fragmento de código. Esta función puede ser increíblemente útil para los programadores, especialmente para aquellos que son nuevos en la programación o que tienen dificultades con un problema en particular. Al simplemente enviar un fragmento de código o un problema a ChatGPT, los usuarios pueden recibir comentarios perspicaces, sugerencias de mejora e incluso código corregido.²⁷

Esta función puede ahorrar tiempo y mejorar la calidad del código al proporcionar una perspectiva fresca y detectar errores que podrían haberse pasado por alto. Además, la capacidad de ChatGPT para proporcionar explicaciones sobre sus revisiones y comentarios puede ayudar a los usuarios a comprender más fácilmente los conceptos de programación y las mejores prácticas.

En general, la capacidad de ChatGPT para revisar, corregir y proporcionar comentarios sobre el código es una herramienta poderosa que puede beneficiar tanto a programadores nuevos como experimentados. Al aprovechar el conocimiento y la experiencia de este modelo de lenguaje de IA, los usuarios pueden mejorar sus habilidades de programación y producir código de mejor calidad.

La siguiente plantilla puede ser útil para este propósito:²⁸

```
A continuación se muestra el código que escribí para resolver el ejercicio:
[insertar aquí el enunciado del ejercicio].
```

Por favor, revisa/mejora/corriges mi código y proporciona comentarios sobre cómo mejorarlo, hacerlo más eficiente o más legible. Además, demuestra cómo habrías codificado la solución de manera diferente, si corresponde.

Código:

[Insertar código aquí]

Pare efectos didácticos, apliquemos esta plantilla al Ejercicio 2:

A continuación se muestra el código que escribí para resolver el ejercicio: "Crea un programa que imprima un triángulo rectángulo utilizando el carácter ''".*

Por favor, revisa/mejora/corriges mi código y proporciona comentarios sobre cómo mejorarlo, hacerlo más eficiente o más legible. Además, demuestra cómo habrías codificado la solución de manera diferente, si corresponde.²⁹

Código:

```
1 print('* ')
2 print('* ' * 2)
3 print('* ' * 3)
4 print('* ' * 4)
5 print('* ' * 5)
```

Aquí tienes una versión mejorada de tu código:

```
1 for i in range(1, 6):
2     print('* ' * i)
```

Este código utiliza un bucle for y la función range para imprimir cada línea del triángulo rectangular. Es más conciso y elimina la necesidad de escribir cada línea por separado. Ten en cuenta que hay muchas formas posibles de resolver este problema, y la mejor solución depende de los requisitos y limitaciones del problema. De todos modos, ¡buen trabajo por tu código inicial! Sigue practicando y aprendiendo.

(NB: ChatGPT tiene razón al sugerir que el uso de un **bucle for** sería un enfoque más elegante y conciso para resolver el problema. Se cubrirán más aspectos sobre los bucles en el Capítulo 4.)

Datos y Variables en Acción2

2.1 Tipos de datos

¿Qué es exactamente un programa? ¿Cómo se relaciona con los datos?³⁰

Un programa es un conjunto de instrucciones que le dice a una computadora qué hacer. Es una secuencia de declaraciones que son ejecutadas por la computadora para realizar una tarea específica. Los programas se escriben en un lenguaje de programación, que es un conjunto de reglas de sintaxis que define cómo deben escribirse tales instrucciones.

Los datos son cualquier tipo de información que puede ser almacenada y procesada por una computadora. Pueden ser números, texto, imágenes, audio o video. Los programas utilizan los datos para realizar sus tareas. Por ejemplo, un programa que calcula el promedio de un conjunto de números necesitará leer los números desde un archivo de datos o pedir al usuario que los ingrese.

La relación entre los programas y los datos es que los programas se pueden utilizar para crear, almacenar, recuperar y manipular datos. También se pueden utilizar para analizar datos y generar nueva información.

Aquí hay algunos ejemplos de cómo se relacionan los programas y los datos:

- ▶ Un procesador de texto utiliza datos (texto) para crear documentos.
- ▶ Una hoja de cálculo utiliza datos (números) para calcular valores.
- ▶ Una base de datos utiliza datos (registros) para almacenar información.
- ▶ Un navegador web utiliza datos (páginas web) para mostrar información en una pantalla.
- ▶ Un juego utiliza datos (personajes, niveles, etc.) para crear una experiencia interactiva.

Entendido. Explica con ejemplos y salidas en pantalla cómo se representa los datos en Python. Proporciona ejemplos de personajes de cómics para entender mejor el concepto.³¹

En Python, los datos se representan asignando valores a variables. Estos valores pueden ser de diferentes tipos, como cadenas de texto, enteros y booleanos. Aquí tienes algunos ejemplos:

```
1 # Asigna una cadena de texto a una variable
2 heroe = "Spider-Man"
3 print("Personaje:", heroe)
4
```

```

5 # Asigna un entero a una variable
6 edad = 23
7 print("Edad:", edad)
8
9 # Asigna un booleano a una variable
10 es_villano = False
11 print("¿Es villano?", es_villano)

```

```

Personaje: Spider-Man
Edad: 23
¿Es villano? False

```

Las variables también pueden recibir valores de diferentes tipos:

```

1 # Asigna una cadena de texto a una variable y luego cambiándola a un entero
2 poder = "Sentido arácnido"
3 print("Poder:", poder)
4
5 poder = 10
6 print("Poder:", poder)
7
8 # Asigna un booleano a una variable y luego cambiándolo a una cadena de texto
9 es_heroe = True
10 print("¿Es héroe?", es_heroe)
11
12 es_heroe = "sí"
13 print("¿Es héroe?", es_heroe) # Salida: sí

```

```

Poder: Sentido arácnido
Poder: 10
¿Es héroe? True
¿Es héroe? sí

```

Python también permite la asignación múltiple en una sola línea, como se muestra a continuación:

```

1 # Asigna múltiples variables a la vez
2 nombre, equipo, universo = "Iron Man", "Avengers", "Marvel"
3 print("-"*10)
4 print("Personaje:", nombre)
5 print("Equipo:", equipo)
6 print("Universo:", universo)

```

```

7 # Actualiza los valores de múltiples variables a la vez
8 nombre, equipo, universo = "Wonder Woman", "Justice League", "DC Comics"
9 print("-"*10)
10 print("Personaje:", nombre)
11 print("Equipo:", equipo)
12 print("Universo:", universo)

```

```

-----
Personaje: Iron Man
Equipo: Avengers
Universo: Marvel
-----
Personaje: Wonder Woman
Equipo: Justice League
Universo: DC Comics

```

En general, puedes pensar en los tipos de datos como la descripción del espacio en memoria que guardará el valor de las variables. Al igual que en la vida real, tenemos diferentes contenedores para diferentes tipos de cosas. Por ejemplo, usamos un vaso para contener líquidos, una caja para contener objetos y una bolsa para contener comestibles. De manera similar, en Python, usamos diferentes tipos de datos para almacenar diferentes instancias de datos.

*Entonces, ¿cómo puedo saber el tipo de una variable en un punto dado del programa?*³²

Puedes utilizar la función incorporada `type()` para determinar el tipo de una variable en Python. Por ejemplo:

```

1 nombre = "Peter Parker"
2 edad = 25
3 altura = 1.83
4 es_spiderman = True
5
6 print("Tipo de nombre:", type(nombre))
7 print("Tipo de edad:", type(edad))
8 print("Tipo de altura:", type(altura))
9 print("Tipo de es_spiderman:", type(es_spiderman))
10
11 # Cambiar el valor (y tipo) de una variable
12 es_spiderman = "sí"
13 print("Tipo de es_spiderman:", type(es_spiderman))

```

```
Tipo de nombre: <class 'str'>
Tipo de edad: <class 'int'>
Tipo de altura: <class 'float'>
Tipo de es_spiderman: <class 'bool'>
Tipo de es_spiderman: <class 'str'>
```

2.2 Entrada/salida básica

¿Cómo puede un programa recibir entrada del usuario de diferentes tipos de datos? Utiliza ejemplos de cómics.³³

Un programa puede obtener entrada del usuario utilizando diferentes funciones integradas dependiendo del tipo de dato a recibir. Aquí tienes algunos ejemplos utilizando personajes de cómics:

1. **Entrada de texto:** La función `input()` se puede utilizar para obtener una entrada de texto del usuario. Por ejemplo, en el cómic de Spider-Man, el programa puede pedir al usuario que ingrese su nombre:

```
1 nombre = input("¿Cuál es tu nombre? ")
2 print("¡Hola, " + nombre + "!")
```

```
¿Cuál es tu nombre? Peter Parker
¡Hola, Peter Parker!
```

2. **Entrada de números enteros:** La función `int()` se puede utilizar para obtener una entrada de números enteros del usuario. Por ejemplo, en el cómic de X-Men, el programa puede pedir al usuario que ingrese su edad:

```
1 edad = int(input("¿Cuántos años tienes? "))
2 if edad >= 18:
3     print("¡Eres lo suficientemente mayor para unirte a los X-Men!")
4 else:
5     print("Lo siento, aún no eres lo suficientemente mayor para unirte a los X-Men.")
6     )
```

```
¿Cuántos años tienes? 20
¡Eres lo suficientemente mayor para unirte a los X-Men!
```

3. **Entrada de números de punto flotante:** La función `float()` se puede utilizar para obtener una entrada de números de punto flotante del usuario. Por ejemplo, en el cómic de Iron Man, el programa puede capturar del usuario el peso de su traje:

```
1 peso = float(input("¿Cuál es el peso de tu traje de Iron Man (en libras)? "))
2 print("Tu traje pesa " + str(peso) + " libras.")
```

```
¿Cuál es el peso de tu traje de Iron Man (en libras)? 500.5
Tu traje pesa 500.5 libras.
```

4. **Entrada booleana:** La función `bool()` se puede utilizar para obtener una entrada booleana del usuario. Por ejemplo, en el cómic de Avengers, el programa puede preguntar al usuario si quiere unirse al equipo:

```
1 respuesta = bool(input("¿Quieres unirme a los Avengers? (True/False) "))
2 if respuesta:
3     print("¡Bienvenido al equipo!")
4 else:
5     print("Quizás la próxima vez.")
```

```
¿Quieres unirme a los Avengers? (True/False) True
¡Bienvenido al equipo!
```

*¿Qué significan esos comandos `if/else` que utilizaste en algunos de los ejemplos anteriores? Puedo imaginar lo que significan, pero ¿puedes explicarlo de manera breve y clara?*³⁴

¡Claro! Una sentencia `if/else` permite que el programa tome decisiones basadas en si una determinada condición es verdadera (`True`) o falsa (`False`). La estructura básica de una sentencia `if/else` es la siguiente:

```
1 if condición:
2     # Código a ejecutar si la condición es verdadera
3 else:
4     # Código a ejecutar si la condición es falsa
```

Observa que los dos bloques de código son mutuamente excluyentes: si la condición es `True`, solo se ejecuta el bloque de código bajo la sentencia `if`, de lo contrario, si la condición es `False`, el bloque bajo la sentencia `else` es el único código que se ejecuta. Por ejemplo, supongamos que queremos escribir un programa que verifique la longitud de una cadena e imprima un mensaje dependiendo de si la cadena es más larga o más corta que un cierto número de caracteres. Podríamos usar una sentencia `if/else` para hacer esto:

```
1 # Definir una cadena
2 heroe = "Mister Fantastic"
3
4 # Verificar la longitud de la cadena
5 if len(heroe) > 10:
6     print("¡El nombre del héroe es realmente largo!")
7 else:
8     print("El nombre del héroe es bastante corto.")
```

En este ejemplo, la función `len()` se utiliza para obtener el número de caracteres (longitud) de la cadena, que en este caso es 15, por lo tanto, el programa imprime el siguiente mensaje:

```
¡El nombre del héroe es realmente largo!
```

Se cubrirá más sobre las estructuras condicionales en el Capítulo 4.

2.3 Un compañero de programación

ChatGPT no solo es capaz de sugerir ejercicios de programación, sino que ¡también puede resolverlos por sí mismo! Esta increíble función abre un mundo completamente nuevo de emocionantes posibilidades de aprendizaje para ti. Puedes desafiarte a ti mismo pidiéndole a ChatGPT que proponga un problema de programación. Una vez que lo hayas resuelto utilizando tus propios conocimientos y habilidades, puedes comparar tu solución con la de ChatGPT. Aquí es donde comienza la verdadera diversión. Puedes obtener comentarios de ChatGPT sobre las diferencias entre tu solución y la suya. Puedes preguntar cuál solución es más eficiente o aprender más sobre nuevas técnicas y trucos que no conocías pero que ChatGPT tiene cubiertos. ¡Imagina cuánto te divertirás con tu nuevo compañero de programación, ChatGPT!

Veamos un ejemplo de cómo funciona esto.

*¡Hola, ChatGPT! Me gustaría practicar mis habilidades de programación con un ejercicio que involucre entrada de usuario, conversión de tipos de datos e impresión de salida, tal vez incluso incluyendo un menú de opciones. ¿Podrías sugerirme un ejercicio en el que pueda trabajar? Una vez que tenga el enunciado, también agradecería si me mostraras un ejemplo de la salida esperada. ¡Gracias!*³⁵

¡Hola, programador en ciernes! Practiquemos tus habilidades de entrada de usuario, conversión de tipos de datos e impresión con un ejercicio entretenido. Escribe un programa que solicite al usuario al menos 5 datos personales, como nombre, edad, hobbies, etc. Luego, ofrece al usuario un menú de

dos opciones: "1-Biografía corta" "2-Biografía larga". Dependiendo de la elección del usuario, tu programa debe imprimir una biografía corta o larga. ¡Sé creativo con la salida y diviértete! Aquí tienes un ejemplo de la salida:

```

¡Bienvenido al Programa de Biografía Personal!

Por favor, ingresa tu nombre: John
Por favor, ingresa tu apellido: Smith
Por favor, ingresa tu edad: 25
Por favor, ingresa tu hobby favorito: libros
Por favor, ingresa tu comida favorita: pizza
Por favor, ingresa tu color favorito: rojo

Menú de opciones:
1. Biografía corta
2. Biografía larga
Ingresa tu elección (1 o 2): 1

--- Biografía corta ---
John, 25 años, es un estudiante de programación cuyo hobby favorito es leer libros.

```

Bien, ahora muéstrame la solución.³⁶

¡Claro! Aquí tienes una posible solución para el Programa de Biografía Personal:

```

1 # Programa de Biografía Personal
2
3 # Solicitar al usuario los datos personales
4 nombre = input("¿Cuál es tu nombre? ")
5 edad = int(input("¿Cuántos años tienes? "))
6 ciudad = input("¿En qué ciudad vives? ")
7 hobby = input("¿Cuál es tu hobby favorito? ")
8 nombre_mascota = input("¿Cuál es el nombre de tu mascota? ")
9
10 # Imprimir el menú de opciones
11 print("Por favor, elige una opción:")
12 print("1. Biografía corta")
13 print("2. Biografía larga")
14
15 # Solicitar al usuario la elección
16 eleccion = input("Ingresa tu elección (1 o 2): ")
17

```

```

18 # Imprimir la biografía según la elección
19 if eleccion == "1":
20     print(f"Hola, mi nombre es {nombre}. Tengo {edad} años y vivo en {ciudad}. Mi hobby
        favorito es {hobby} y tengo una mascota llamada {nombre_mascota}.")
21 elif eleccion == "2":
22     print(f"Mi nombre es {nombre} y tengo {edad} años. Actualmente vivo en {ciudad} y en
        mi tiempo libre disfruto {hobby}. Tengo una mascota llamada {nombre_mascota} que me
        hace compañía. ¡Gracias por leer!")
23 else:
24     print("Opción inválida. Por favor, ingresa 1 o 2.")

```

*Genial. Mi versión de la solución se muestra a continuación. ¿Qué piensas al respecto? ¿En comparación con la tuya?*³⁷

```

nombre = input("¿Cuál es tu nombre? ")
edad = int(input("¿Cuántos años tienes? "))
ciudad = input("¿En qué ciudad vives? ")
hobby = input("¿Cuál es tu hobby favorito? ")
comida = input("¿Cuál es tu comida favorita? ")
color = input("¿Cuál es tu color favorito? ")
tarjeta_credito = input("¿Cuál es el número de tu tarjeta de crédito? ")

print('+ '+'-'*20+'+')
print(f"|{'OPCIONES':^20}|")
print('+ '+'-'*20+'+')
print(f"|{'1. Biografía corta':^20}|")
print(f"|{'2. Biografía larga':^20}|")
print('+ '+'-'*20+'+')

opcion = int(input("Ingresa el número de la opción: "))

if opcion == 1:
    print(f"\n----- Biografía personal de: {nombre} -----")
    print(f"{nombre} tiene {edad} años y vive en {ciudad}.")
    print("-----")
else:
    print(f"\n----- Biografía personal de: {nombre}
        -----")
    print(f"{nombre} tiene {edad} años y vive en {ciudad}.")
    print(f"En el tiempo libre de {nombre}, le gusta {hobby} y comer {comida}.")

```



```
print(f"Un dato curioso sobre {nombre} es que su color favorito es {color}.")
print(f"A modo de información, el número de tarjeta de crédito de {nombre}
es: '+'*'*len(tarjeta_credito) )
print("-----")
```

Tu solución se ve bien y cumple la misma tarea que la solución que proporcioné. Una diferencia que noté es que agregaste una capa adicional de formato a las opciones del menú utilizando los caracteres '+' y '-' para crear un borde alrededor del menú. Esto añade un toque visual atractivo al programa. Además, tu programa también utiliza f-strings para dar formato a la salida, que es una forma concisa y elegante de mostrar información. Lo único que te recomendaría es evitar solicitar información sensible como números de tarjeta de crédito en un programa como este, especialmente dado que el programa lo imprime parcialmente oculto. Es importante priorizar la seguridad y privacidad del usuario en todas las aplicaciones. En general, ¡buen trabajo!

Pero me aseguré de ocultar el número de tarjeta de crédito. Solo estaba bromeando.³⁸

Ah, ¡entiendo! Lamento la confusión. En ese caso, tu código se ve genial y funciona correctamente. Siempre es una buena idea tener cuidado con la información sensible, incluso en ejemplos de código. En general, tu implementación es concisa y legible, y utiliza de manera efectiva las opciones de formato de cadena disponibles en Python para que la salida se vea ordenada y organizada. ¡Bien hecho!

Por cierto, me gusta cómo comentaste cada sección de tu código; ayuda a aclarar las cosas. Por favor, haz lo mismo con mi código.³⁹

Claro, aquí tienes tu código con comentarios:

```
1 # Solicitar al usuario la entrada de datos personales
2 nombre = input("¿Cuál es tu nombre? ")
3 edad = int(input("¿Cuántos años tienes? "))
4 ciudad = input("¿En qué ciudad vives? ")
5 hobby = input("¿Cuál es tu hobby favorito? ")
6 comida = input("¿Cuál es tu comida favorita? ")
7 color = input("¿Cuál es tu color favorito? ")
8 tarjeta_credito = input("¿Cuál es tu número de tarjeta de crédito? ")
9
10 # Imprimir las opciones del menú
11 print('+ '+'-'*20+'+')
12 print(f"|{'OPCIONES':^20}|")
13 print('+ '+'-'*20+'+')
14 print(f"|{'1. Biografía corta':^20}|")
15 print(f"|{'2. Biografía larga':^20}|")
16 print('+ '+'-'*20+'+')
```

```

17
18 # Solicitar al usuario que elija una opción
19 eleccion = int(input("Ingresa el número de la opción: "))
20
21 # Imprimir la biografía personal elegida
22 if eleccion == 1:
23     print(f"\n----- Biografía personal de: {nombre} -----")
24     print(f"{nombre} tiene {edad} años y vive en {ciudad}.")
25     print("-----")
26 else:
27     print(f"\n----- Biografía personal de: {nombre} -----")
28     print(f"{nombre} tiene {edad} años y vive en {ciudad}.")
29     print(f"En el tiempo libre de {nombre}, le gusta {hobby} y comer {comida}.")
30     print(f"Un dato curioso sobre {nombre} es que su color favorito es {color}.")
31     print(f"A modo de información, el número de tarjeta de crédito de {nombre} es: '+'*'*
32         len(tarjeta_credito) )
33     print("-----")

```

2.4 ¡Manos a la obra!

Proporciona cuatro ejercicios de código relacionados con la entrada y conversión de tipos de datos que necesito corregir para que se ejecuten correctamente, ajustando errores de sintaxis o lógica.⁴⁰

Claro, aquí tienes cuatro ejercicios de código relacionados con la entrada y conversión de tipos de datos, con errores de sintaxis y/o lógica que necesitan ser corregidos:

Ejercicio 1:

```

1 edad = input("Ingresa tu edad: ")
2 if edad >= 18:
3     print("¡Eres lo suficientemente mayor para votar!")
4 else:
5     print("Aún no eres lo suficientemente mayor para votar.")

```

Ejercicio 2:

```

1 numero = input("Ingresa un número decimal: ")
2 numero = bool(numero)
3 print(f"La parte entera del {numero} es {numero}")

```

Ejercicio 3:

```

1 password = input("Ingresa tu contraseña: ")
2 if length(password) >= 8:
3     print("¡Tu contraseña es segura!")
4 else:
5     print("Tu contraseña es demasiado corta. Por favor, elige una contraseña de al menos
    8 caracteres.")

```

Ejercicio 4:

```

1 nombre = input("Ingresa tu nombre: ")
2 edad = input("Ingresa tu edad: ")
3 print("¡Hola, nombre! Tienes edad años.")

```

Sugiere algunos ejercicios que requieran escribir código para resolverlos, utilizando solo la entrada del usuario y operaciones de conversión de tipos de datos.⁴¹

Ejercicio 1:

Escribe un programa que solicite al usuario que ingrese su año de nacimiento y calcule su edad. A continuación se muestra un ejemplo de la salida por pantalla.

```

Ingresa tu nombre: Tony Stark
Ingresa tu año de nacimiento: 1970

¡Hola Tony Stark! En 2023, tienes 53 años.

```

Ejercicio 2:

Escribe un programa que solicite al usuario ingresar una temperatura en grados Celsius y la convierta a grados Fahrenheit. La fórmula de conversión es $F = (C * 9/5) + 32$. A continuación se muestra un ejemplo de la salida por pantalla.

```

Ingresa la temperatura en grados Celsius: 30
30 grados Celsius es igual a 86 grados Fahrenheit.

```

(NB: Este ejercicio requiere el uso de operadores aritméticos, incluyendo suma, multiplicación y división. Más sobre estos y otros operadores matemáticos se cubrirá en el Capítulo 3).

Ejercicio 3:

Crea un programa calculadora que solicite al usuario que ingrese dos números y seleccione una operación para realizar con ellos. El programa debe mostrar un menú de opciones de operaciones (suma, resta, multiplicación y división) y pedir al usuario que seleccione el

número correspondiente a la operación deseada. Una vez que el usuario haya ingresado su opción, el programa debe mostrar el resultado de la operación seleccionada. A continuación se muestra un ejemplo de la salida por pantalla.

```
¡Bienvenido a la aplicación de calculadora!
```

```
Ingresa el primer número: 10
```

```
Ingresa el segundo número: 5
```

```
----- Menú de operaciones -----
```

```
1. Suma
```

```
2. Resta
```

```
3. Multiplicación
```

```
4. División
```

```
Ingresa el número de la operación: 3
```

```
10 * 5 = 50
```

(NB: Nuevamente, se cubrirá más sobre estos y otros operadores matemáticos en el Capítulo 3).

La Sala de Operaciones3

Enumera todos los operadores de Python y proporciona una breve descripción para cada uno.⁴²

Aquí tienes una lista de los diferentes tipos de operadores en Python y sus definiciones:

1. **Operadores aritméticos:** se utilizan para realizar operaciones matemáticas, como la suma, resta, multiplicación, división, etc.
2. **Operadores de comparación:** se utilizan para comparar valores y devolver un valor booleano (Verdadero o Falso), como igual a, no igual a, mayor que, menor que, etc.
3. **Operadores lógicos:** se utilizan para evaluar la verdad o falsedad de combinaciones de proposiciones o condiciones.
4. **Operadores a nivel de bits:** se utilizan para realizar operaciones en bits individuales de números binarios.
5. **Operadores de asignación:** se utilizan para asignar valores a variables, incluso operándolas con operadores aritméticos.
6. **Operadores de identidad:** se utilizan para comparar la ubicación en memoria de dos objetos.
7. **Operadores de pertenencia:** se utilizan para comprobar si un valor pertenece o no pertenece en una secuencia.
8. **Operador de exponenciación:** se utiliza para elevar un número a una potencia.

Así que demos un vistazo más de cerca a cada uno de ellos.

3.1 Operadores aritméticos

Explica los detalles de los operadores aritméticos en Python en un tono divertido e incluye un ejemplo de código para ilustrar el uso de todos los operadores. El ejemplo debe comparar estadísticas entretenidas sobre las películas de Marvel y X-Men y su recaudo en taquilla.⁴³

En Python, los operadores aritméticos se utilizan para realizar operaciones matemáticas básicas, como la suma, resta, multiplicación y división. Estos operadores se utilizan para manipular tipos de datos numéricos como enteros y números de punto flotante.

Seguidamente se presentan los operadores aritméticos clave en Python:

- **Suma:** El signo más (+) se utiliza para la suma de dos valores. Suma el segundo valor al primero. Por ejemplo, $2 + 2$ es igual a 4.
- **Resta:** El signo menos (-) se utiliza para la resta. Resta el segundo valor del primer valor. Por ejemplo, $5 - 2$ es igual a 3.

- **Multiplicación:** El asterisco (*) se utiliza para la multiplicación. Multiplica dos valores. Por ejemplo, $3 * 4$ es igual a 12.
- **División:** La barra diagonal (/) se utiliza para la división. Divide el primer valor por el segundo valor. Por ejemplo, $8 / 2$ es igual a 4.
- **División entera:** La doble barra diagonal (//) se utiliza para la división entera. Divide el primer valor por el segundo valor, pero redondea hacia abajo al número entero más cercano. Por ejemplo, $8 // 3$ es igual a 2.
- **Módulo:** El signo de porcentaje (%) se utiliza para el módulo. Devuelve el resto de la división del primer valor por el segundo valor. Por ejemplo, $7 \% 3$ es igual a 1.

Divirtámonos un poco con estos operadores. Aquí tienes un ejemplo que compara algunas estadísticas sobre las películas de Marvel y X-Men y su taquilla:

```

1 # Estadísticas de películas de Marvel y X-Men
2 marvel_peliculas = 28
3 xmen_peliculas = 13
4 marvel_taquilla = 22.55e9 # en dólares estadounidenses
5 xmen_taquilla = 6.01e9    # en dólares estadounidenses
6 boleto_individual = 15    # en dólares estadounidenses
7
8 # Suma
9 total_peliculas = marvel_peliculas + xmen_peliculas
10 print("Número total de películas de ambas franquicias:", total_peliculas)
11
12 # Resta
13 diferencia_peliculas = marvel_peliculas - xmen_peliculas
14 print("Marvel tiene", diferencia_peliculas, "películas más que X-Men")
15
16 # Multiplicación
17 dinero_necesario = boleto_individual * total_peliculas
18 print("Costo total de ver todas las películas en el cine: US$", dinero_necesario)
19
20 # División
21 taquilla_total = marvel_taquilla + xmen_taquilla
22 taquilla_promedio = taquilla_total / total_peliculas
23 print(f"Promedio de taquilla por película: US$ {taquilla_promedio:,.2f}")
24
25 # División entera
26 años_necesarios = total_peliculas // 12 # asumiendo ver 1 película al mes
27
28

```

```

29 # Módulo
30 meses_extra = total_peliculas % 12 # asumiendo ver 1 película al mes
31
32 print(f"Tiempo necesario para ver todas las películas (una al mes): {años_necesarios} añ
    os y {meses_extra} meses.")

```

El código es autoexplicativo y produce la siguiente salida:

```

Número total de películas de ambas franquicias: 41
Marvel tiene 15 películas más que X-Men
Costo total de ver todas las películas en el cine: US$ 615
Promedio de taquilla por película: US$ 696,585,365.85
Tiempo necesario para ver todas las películas (una al mes): 3 años y 5 meses.

```

¿Qué significan esos números 22.55e9 y 6.01e9 para las variables de taquilla?⁴⁴

Los números 22.55e9 y 6.01e9 están en notación científica, que es una forma de expresar números muy grandes o muy pequeños utilizando potencias de 10. En este caso, **e9** significa “por 10 elevado a la 9ª potencia”, es decir, mil millones. Por lo tanto, 22.55e9 significa 22.55 mil millones, y 6.01e9 significa 6.01 mil millones. Estos números probablemente representan valores aproximados de recaudo de taquilla en dólares estadounidenses de las películas de Marvel y X-Men, respectivamente.

3.2 Operadores de comparación

Explica los detalles de los operadores de comparación en Python con un tono divertido e incluye un ejemplo de código para ilustrar el uso de todos ellos, esta vez utilizando datos curiosos sobre equipos de fútbol.⁴⁵

Los operadores de comparación, también llamados operadores relacionales, se utilizan para comparar valores en Python. El resultado de la comparación siempre es un valor booleano: ya sea **True** o **False**.

Aquí tienes una lista de los operadores de comparación en Python:

- **== (igual a):** Comprueba si los valores de los dos operandos son iguales.
- **!= (diferente de):** Comprueba si los valores de los dos operandos no son iguales.
- **< (menor que):** Comprueba si el valor del operando izquierdo es menor que el valor del operando derecho.
- **<= (menor o igual que):** Comprueba si el valor del operando izquierdo es menor o igual al valor del operando derecho.

- **> (mayor que):** Comprueba si el valor del operando izquierdo es mayor que el valor del operando derecho.
- **>= (mayor o igual que):** Comprueba si el valor del operando izquierdo es mayor o igual al valor del operando derecho.

Ilustremos el uso de todos estos operadores con un ejemplo divertido sobre equipos de fútbol. Imagina que tenemos dos equipos, Real Madrid y Barcelona, y queremos comparar algunas de sus estadísticas:

```
1 # Estadísticas del Real Madrid
2 puntos_real = 70
3 goles_ anotados_real = 76
4 goles_recibidos_real = 39
5 diferencia_goles_real = goles_ anotados_real - goles_recibidos_real
6
7
8 # Estadísticas del Barcelona
9 puntos_barca = 65
10 goles_ anotados_barca = 85
11 goles_recibidos_barca = 38
12 diferencia_goles_barca = goles_ anotados_barca - goles_recibidos_barca
13
14
15 # Operadores de comparación
16 print(f"Real Madrid ({puntos_real}) y Barcelona ({puntos_barca}) tienen la misma cantidad
    de puntos:",
17       puntos_real == puntos_barca)
18 print(f"Real Madrid ({goles_ anotados_real}) y Barcelona ({goles_ anotados_barca}) tienen
    una cantidad diferente de goles anotados:",
19       goles_ anotados_real != goles_ anotados_barca)
20 print(f"Real Madrid ({goles_recibidos_real}) ha recibido menos goles que Barcelona ({
    goles_recibidos_barca}):",
21       goles_recibidos_real < goles_recibidos_barca)
22 print(f"La diferencia de goles del Real Madrid ({diferencia_goles_real}) es menor o igual
    a la diferencia de goles del Barcelona ({diferencia_goles_barca}):",
23       diferencia_goles_real <= diferencia_goles_barca)
24 print(f"Real Madrid ({puntos_real}) tiene más puntos que Barcelona ({puntos_barca}):",
25       puntos_real > puntos_barca)
26 print(f"Barcelona ({goles_recibidos_barca}) ha recibido más goles que el Real Madrid ({
    goles_recibidos_real}):",
27       goles_recibidos_barca >= goles_recibidos_real)
```



```

Real Madrid (70) y Barcelona (65) tienen la misma cantidad de puntos: False

Real Madrid (76) y Barcelona (85) tienen una cantidad diferente de goles anotados:
    True

Real Madrid (39) ha recibido menos goles que Barcelona (38): False

La diferencia de goles del Real Madrid (37) es menor o igual a la diferencia de
    goles del Barcelona (47): True

Real Madrid (70) tiene más puntos que Barcelona (65): True

Barcelona (38) ha recibido más goles que el Real Madrid (39): False

```

Como ilustra este ejemplo, estos operadores de comparación son muy útiles en la programación de Python, ya que nos permiten verificar las relaciones entre diferentes valores y tomar decisiones basadas en esas relaciones.

3.3 Operadores lógicos

Explica el concepto de los operadores lógicos en Python en un tono informal. Después de definir cada operador, incluye un ejemplo de código para ilustrar el uso de todos ellos. Para este fin, usa una tabla de platos de comida rápida indicando su idoneidad para vegetarianos y veganos.⁴⁶

Los operadores lógicos en Python son herramientas que nos permiten combinar y manipular condiciones booleanas que representan afirmaciones verdaderas o falsas. Estos operadores se utilizan principalmente con variables booleanas, que solo pueden ser **True** o **False**.

Los operadores lógicos más comúnmente utilizados en Python son **and** (y), **or** (o) y **not** (no). El operador **and** devuelve **True** solo si ambos operandos son **True**, mientras que el operador **or** devuelve **True** si al menos uno de los operandos es **True**. El operador **not** se utiliza para invertir el valor booleano de su operando.

Consideremos la siguiente tabla de platos de comida rápida que indica su idoneidad para vegetarianos y veganos:

Plato	¿Vegetariano?	¿Vegano?
Hamburguesa con queso	No	No
Pizza Margherita	Sí	No
Ensalada de jardín	Sí	Sí
Helado	Sí	No

Podemos usar operadores lógicos para responder preguntas como “¿Qué platos son adecuados tanto para vegetarianos como para veganos?” o “¿Qué platos no son adecuados ni para vegetarianos ni para veganos?” Aquí tienes un ejemplo de código para demostrar el uso de los operadores lógicos:

```

1 # Definimos los valores booleanos de la tabla de comidas rápidas
2 # (cb: Hamburguesa con queso, mp: Pizza Margherita, ej: Ensalada de jardín, ic: Helado)
3 cb_veg = False
4 cb_vegan = False
5 mp_veg = True
6 mp_vegan = False
7 ej_veg = True
8 ej_vegan = True
9 ic_veg = True
10 ic_vegan = False
11
12 # Imprimimos los resultados de los operadores lógicos
13 print("¿La Hamburguesa con queso es vegetariana y vegana? ", cb_veg and cb_vegan)
14 print("¿La Pizza Margherita es vegetariana y vegana? ", mp_veg and mp_vegan)
15 print("¿La Ensalada de jardín es vegetariana y vegana? ", ej_veg and ej_vegan)
16 print("¿El Helado es vegetariano y vegano? ", ic_veg and ic_vegan)
17 print("¿La Hamburguesa con queso es vegetariana o vegana? ", cb_veg or cb_vegan)
18 print("¿La Pizza Margherita es vegetariana o vegana? ", mp_veg or mp_vegan)
19 print("¿La Ensalada de jardín es vegetariana o vegana? ", ej_veg or ej_vegan)
20 print("¿El Helado es vegetariano o vegano? ", ic_veg or ic_vegan)
21 print("¿La Hamburguesa con queso no es vegetariana? ", not cb_veg)
22 print("¿La Hamburguesa con queso no es vegana? ", not cb_vegan)

```

La salida se muestra a continuación:

```

¿La Hamburguesa con queso es vegetariana y vegana? False
¿La Pizza Margherita es vegetariana y vegana? False
¿La Ensalada de jardín es vegetariana y vegana? True
¿El Helado es vegetariano y vegano? False
¿La Hamburguesa con queso es vegetariana o vegana? False
¿La Pizza Margherita es vegetariana o vegana? True
¿La Ensalada de jardín es vegetariana o vegana? True
¿El Helado es vegetariano o vegano? True
¿La Hamburguesa con queso no es vegetariana? True
¿La Hamburguesa con queso no es vegana? True

```

3.4 Operadores a nivel de bit

Explica el concepto de los operadores a nivel de bit en Python de forma amigable pero profesional. Incluye un ejemplo de código después de definir cada operador para demostrar cómo usarlos. Muestra los resultados de las operaciones, incluyendo las secuencias de bits de los operandos y sus correspondientes representaciones en decimal.⁴⁷

Los operadores bit a bit en Python se utilizan para manipular los bits o dígitos binarios de un número entero. Se aplican a cada posición de bit de un entero de forma independiente, de ahí el nombre de “bit a bit”.

Hay seis operadores bit a bit en Python: AND (&), OR (|), XOR (^), NOT (~), desplazamiento a la izquierda (<<) y desplazamiento a la derecha (>>). Veámos:

- El operador AND bit a bit devuelve un nuevo entero donde cada bit se establece en 1 solo si ambos bits de los operandos enteros originales son 1. De lo contrario, el bit se establece en 0.

Aquí tienes un ejemplo:

```
1 a = 12 # 1100 en binario
2 b = 25 # 11001 en binario
3 c = a & b # 1000 en binario
4 print(f" {a:08b} (decimal: {a})\n & {b:08b} (decimal: {b})\n  -----\n {c:08b} (decimal: {c})")
5
```

Observa que utilicé el modificador de formato :08b para imprimir los enteros como cadenas binarias de 8 bits para mayor claridad. La salida sería:

```
00001100 (decimal: 12)
& 00011001 (decimal: 25)
-----
00001000 (decimal: 8)
```

- El operador OR bit a bit devuelve un nuevo entero donde cada bit se establece en 1 si al menos uno de los bits de los operandos enteros originales es 1. De lo contrario, el bit se establece en 0. Ejemplo:

```
1 a = 12 # 1100 en binario
2 b = 25 # 11001 en binario
3 c = a | b # 11001 en binario
4 print(f" {a:08b} (decimal: {a})\n | {b:08b} (decimal: {b})\n  -----\n {c:08b} (decimal: {c})")
5
```

La salida sería:

```
00001100 (decimal: 12)
| 00011001 (decimal: 25)
-----
00011101 (decimal: 29)
```

- El operador XOR bit a bit devuelve un nuevo entero donde cada bit se establece en 1 solo si uno de los bits de los operandos enteros originales es 1. De lo contrario, el bit se establece en 0. Ejemplo:

```
1 a = 12 # 1100 en binario
2 b = 25 # 11001 en binario
3 c = a ^ b # 10001 en binario
4 print(f" {a:08b} (decimal: {a})\n ^ {b:08b} (decimal: {b})\n  -----\n  {c:08b} (decimal: {c})")
5
```

La salida sería:

```
00001100 (decimal: 12)
^ 00011001 (decimal: 25)
-----
00010101 (decimal: 21)
```

- El operador NOT bit a bit devuelve el complemento de un entero, es decir, cambia cada bit de 0 a 1 y viceversa. Ten en cuenta que devuelve el resultado en una forma de complemento a dos, que es una representación con signo de enteros. Ejemplo:

```
1 a = 12 # 1100 en binario
2 c = ~a # -13 en decimal (complemento de 1100 en binario)
3 print(f" ~ {a:08b} (decimal: {a})\n  -----\n  {c:08b} (decimal: {c})")
4
```

La salida sería:

```
~ 00001100 (decimal: 12)
-----
-0001101 (decimal: -13)
```

- El operador de desplazamiento a la izquierda mueve todos los bits de un entero hacia la izquierda un número dado de posiciones y rellena los bits más a la derecha

con ceros. Esto es equivalente a multiplicar el entero por 2 elevado a la potencia del número de posiciones. Ejemplo:

```
1 a = 12 # 1100 en binario
2 c = a << 3 # 110000 en binario, que es 96 en decimal
3 print(f" {a:08b} (decimal: {a})\n<< 3 \n  ----- \n {c:08b} (decimal: {c})")
4
```

La salida sería:

```
00001100 (decimal: 12)
<< 3
-----
01100000 (decimal: 96)
```

- El operador de desplazamiento a la derecha mueve todos los bits de un entero hacia la derecha un número dado de posiciones y rellena los bits más a la izquierda con ceros (para enteros no negativos) o unos (para enteros negativos). Esto es equivalente a dividir el entero por 2 elevado a la potencia del número de posiciones. Ejemplo:

```
1 a = 12 # 1100 en binario
2 c = a >> 3 # 11 en binario, que es 1 en decimal
3 print(f" {a:08b} (decimal: {a})\n>> 3 \n  ----- \n {c:08b} (decimal: {c})")
4
```

La salida sería:

```
00001100 (decimal: 12)
>> 3
-----
00000001 (decimal: 1)
```

Los operadores bit a bit se utilizan principalmente en programación de bajo nivel, como sistemas embebidos o controladores de dispositivos, pero también se pueden utilizar en criptografía, procesamiento de gráficos y compresión de datos.

De acuerdo, pero ¿podrías explicar brevemente a qué te refieres con “dígitos binarios de un entero”? Explícalo con algunas analogías de la vida real que sean divertidas.⁴⁸

En los sistemas informáticos, toda la información se representa en términos de dígitos binarios o bits. Estos son los 0 y 1 que tal vez hayas visto en código informático o electrónica digital. Un dígito binario es un *bit*, que es la unidad más pequeña de datos que una computadora puede almacenar o manipular.

Un bit solo puede tener dos posibles valores, 0 o 1, que equivalentemente puede representar estados como **False** o **True**, o, prendido o apagado, o, bajo o alto, o, cara o sello, etc. Una secuencia de 8 bits consecutivos se conoce como *byte*.

Una analogía entretenida para comprender los bits y los bytes podría ser la de un grupo de interruptores de luz. Cada interruptor puede estar encendido o apagado, al igual que un solo bit puede ser 0 o 1. Al colocar varios interruptores uno al lado del otro, podemos formar grupos más grandes de interruptores y representar valores más grandes.

Por ejemplo, 4 interruptores pueden tener 16 combinaciones diferentes de bombillos prendidos o apagados, que podrían representar valores desde 0 hasta 15. Del mismo modo, 8 interruptores pueden tener 256 combinaciones diferentes de prendido/apagado que pueden representar valores desde 0 hasta 255. Por ejemplo, el número 6 se puede representar en una secuencia binaria de 4 bits como 0110, lo que significa $0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$, esto es, $0 + 4 + 2 + 0$. De manera similar, el número 15 sería 1111, lo que significa $1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$, esto es, $8 + 4 + 2 + 1$. Entonces, cuando hablamos de los dígitos binarios de un entero, nos referimos a la representación de ese entero en notación binaria.

Los operadores bit a bit funcionan manipulando estos interruptores o bits individuales, así como grupos de ellos. Esto nos permite realizar una variedad de operaciones útiles en los datos subyacentes, como desplazarlos a la izquierda o a la derecha, invertir bits individuales o combinar múltiples valores de varias formas.

De acuerdo, una última cosa: por favor, explícame qué quieres decir con “representación en forma de complemento a dos” de manera breve y amigable. ⁴⁹

En los sistemas informáticos, a menudo necesitamos representar números negativos, así como números positivos. Una forma de representar números negativos es utilizar la representación de signo-magnitud, donde el bit más a la izquierda representa el signo del número (0 para positivo, 1 para negativo) y los bits restantes representan la magnitud. Sin embargo, esta representación tiene algunas desventajas, como la necesidad de casos especiales para manejar el cero y el hecho de que tenemos dos representaciones del cero (positivo y negativo).

La representación en forma de complemento a dos es una alternativa más eficiente en memoria y ampliamente utilizada. Representa los números negativos como el complemento a dos de su valor absoluto. Esto significa que, para obtener el complemento a dos de un número, primero tomamos el complemento (invertimos todos los bits) de la representación binaria de su valor absoluto y luego le sumamos 1 al resultado. Veamos un ejemplo:

1. Encuentra la representación binaria de 5: 0000101.
2. Invierte todos los bits: 11111010.
3. Suma 1: 11111011.

Entonces, la representación en forma de complemento a dos de -5 en binario de 8 bits es 11111011.

Los números negativos en complemento a dos son útiles en operaciones aritméticas porque permiten a la computadora realizar sumas y restas utilizando el mismo circuito y algoritmos que para los enteros sin signo. De hecho, sumar un número negativo en complemento a dos es equivalente a restar su valor absoluto del entero sin signo. Esto simplifica la implementación de las operaciones aritméticas y reduce la cantidad de hardware requerido.

En resumen, la representación en forma de complemento a dos proporciona una forma simple y eficiente de representar números negativos en sistemas informáticos, lo cual es esencial para muchas aplicaciones, como el procesamiento digital de señales, la inteligencia artificial y la criptografía.

3.5 Operadores de asignación

Explica el concepto de los operadores de asignación aritmética en Python de manera divertida.⁵⁰

Los operadores de asignación aritmética en Python son simplemente una forma divertida de combinar matemáticas con un atajo. ¿Sabes que a veces quieres actualizar un valor total sumando o restando otro valor sin usar otra variable para calcular el valor temporal? Bueno, ¡los operadores de asignación aritmética están aquí para facilitarte la codificación!

Permíteme darte algunos ejemplos de la vida real. Imagina que tienes un frasco lleno de dulces. Quieres llevar un registro de cuántos dulces te quedan disponibles cada día. Comienzas con 10 dulces y en el primer día te comes 3. En lugar de calcular la nueva cantidad de dulces disponibles manualmente, puedes usar el operador de asignación aritmética `-=` para restar los 3 que te comiste y dejar que el operador te actualice el nuevo total disponible en el frasco. Es decir en vez de calcular `dulces_restantes = dulces - 3` y luego actualizar `dulces = dulces_restantes`, puedes simplemente escribir `dulces -= 3` y, ya está, ¡mataste dos pájaros con un solo tiro!

De manera similar, supongamos que tienes una alcancía llena de monedas. Imagina que esta alcancía es un tipo de alcancía digital moderna que te permite obtener un 10 % de interés sobre el dinero que ahorras cada mes. Comienzas con \$10 y al final del mes tienes \$11. En lugar de asignar manualmente el valor final de la alcancía, usa el operador de asignación aritmética `*=` para agregar 1 al saldo actual. En lugar de escribir `alcancia = 11`, puedes escribir `alcancia *= 1.1`, así:

```
1 alcancia = 10
2 print("Dinero en la alcancía ahora:\t$ ", alcancia)
3 alcancia *= 1.1
4 print("Dinero después de un mes:\t$ ", alcancia)
```

Dinero en la alcancía ahora: \$ 10
 Dinero después de un mes: \$ 11.0

Aquí tienes algunos otros ejemplos de operadores de asignación aritmética en Python:

```

1 x = 10
2 x += 5 # equivalente a x = x + 5
3 print(x) # Salida: 15
4
5 y = 20
6 y -= 3 # equivalente a y = y - 3
7 print(y) # Salida: 17
8
9 z = 7
10 z *= 2 # equivalente a z = z * 2
11 print(z) # Salida: 14
12
13 a = 16
14 a /= 4 # equivalente a a = a / 4
15 print(a) # Salida: 4.0
16
17 b = 5
18 b %= 2 # equivalente a b = b % 2
19 print(b) # Salida: 1

```

Como puedes ver, los operadores de asignación aritmética son una forma divertida y eficiente de actualizar los valores de las variables en Python.

3.6 Operadores de identidad

Explica el concepto de los operadores de identidad en Python de manera divertida.⁵¹

Básicamente, los operadores de identidad se utilizan para comparar las ubicaciones de memoria de dos variables en Python. Devuelven un valor booleano de **True** o **False**, indicando si las variables son iguales o diferentes en términos de su ubicación de memoria.

Para entenderlo mejor, imagina que tienes dos tazas idénticas. A simple vista, pueden parecer exactamente iguales, pero si te fijas más de cerca, notarás que tienen diferentes huellas dactilares. En Python, los operadores de identidad serían como verificar si dos objetos tienen la misma huella dactilar o no.

Aquí están los dos operadores de identidad en Python:

- **is**: devuelve **True** si dos variables se refieren al mismo objeto en memoria, y **False** en caso contrario.
- **is not**: devuelve **True** si dos variables no se refieren al mismo objeto en memoria, y **False** en caso contrario.

Veamos un ejemplo. En este código, definimos variables con los nombres de personajes de la película *Cars*. Luego usamos los operadores de identidad para compararlos:

```

1 # Definimos los autos
2 mcqueen = "Rayo McQueen"
3 doc = "Doc Hudson"
4 mater = "Mate"
5
6 # Usamos los operadores de identidad para comparar los autos
7 print(f"{mcqueen} es {doc}: ", mcqueen is doc)
8 print(f"{mcqueen} no es {mater}: ", mcqueen is not mater)
9
10 # Asignamos una variable a la otra
11 mcqueen = doc
12 print(f"Después de asignar, {mcqueen} es {doc}: ", mcqueen is doc)
13
14 # Verificamos si un auto es None
15 sally = None
16 print(f"{sally} es None: ", sally is None)
17
18 sally = "Sally Carrera"
19 print(f"Después de asignar, {sally} es None: ", sally is None)

```

```

Rayo McQueen es Doc Hudson: False
Rayo McQueen no es Mate: True
Después de asignar, Doc Hudson es Doc Hudson: True
None es None: True
Después de asignar, Sally Carrera es None: False

```

En el último bloque, utilizamos el operador **is** para verificar si el objeto al que se refiere la variable **sally** es el objeto especial de Python **None**. Inicialmente, el resultado será **True**, ya que la variable no ha recibido ningún nombre, pero después de asignar un nombre, el resultado será **False**.

Entonces, los operadores de identidad en Python te permiten comparar dos objetos y descubrir si son iguales o no, sin necesidad de mirar sus valores.

Vale. Dos preguntas. Primero, ¿a qué te refieres exactamente con “ubicaciones en la memoria”? No veo ninguna ubicación de memoria en el código; por favor, acláralo de manera amigable y colorida. Segundo, ¿qué es la variable especial de Python “None”? Por favor, acláralo.⁵²

Claro, las ubicaciones de memoria en programación son como unidades de almacenamiento donde se guarda la información en la memoria de una computadora. Imagínalo como una gran cuadrícula de casillas postales, donde cada casilla es un espacio de memoria ubicado en una fila y columna determinada de la cuadrícula, y cada casilla puede almacenar una pieza de información. Entonces, cuando creamos una variable, básicamente estamos reservando uno de estos espacios de memoria para almacenar algún dato, como un número, una cadena o incluso otro objeto. Cuando asignamos un valor a esa variable, estamos almacenando ese valor en el espacio de memoria asignado a esa variable.

En cuanto a tu segunda pregunta, en Python, `None` es un valor especial que representa la ausencia de un valor. Se utiliza para indicar que una variable no tiene ningún valor o que una función no devuelve nada. Imagínalo como una caja vacía en tu unidad de almacenamiento; sigue siendo una caja, pero no hay nada dentro de ella. Cuando asignas una variable a `None`, básicamente estás reservando una ubicación de memoria para esa variable, pero no asignas ningún valor real a ella.

3.7 Operadores de pertenencia

Explica el concepto de los operadores de pertenencia en Python de manera lúdica.⁵³

Los operadores de pertenencia en Python son como detectives secretos que nos ayudan a investigar si un valor en particular pertenece a un grupo específico o no.

El primer operador de pertenencia es `in`, que verifica si un valor específico (como una letra o un nombre) está presente en una cadena más grande o no. Es como preguntarle al portero de una discoteca si tu nombre está en la lista de invitados o no. Por ejemplo:

```
1 nombre = "Alicia"
2 otro_nombre = "Bruno"
3 letra = 'A'
4 print(f"{nombre} comienza con {letra}: ", letra in nombre)
5 print(f"{otro_nombre} comienza con {letra}: ", letra in otro_nombre)
```

```
Alicia comienza con A: True
Bruno comienza con A: False
```

En este ejemplo, el código verifica si la letra 'A' es la primera letra de alguno de los nombres (asumiendo que solo la primera letra de un nombre está en mayúscula).

El segundo operador de pertenencia es `not in`, que verifica si un valor dado no está presente en una cadena más grande o no. El valor puede ser una letra o una secuencia de letras. Por ejemplo:

```
1 frase1 = "¡Java es divertido!"
2 frase2 = "¡Java es divertido, pero Python es realmente fantástico!"
3 lenguaje = "Python"
4
5 print(f"'{lenguaje}' no está en '{frase1}': ", lenguaje not in frase1)
6 print(f"'{lenguaje}' no está en '{frase2}': ", lenguaje not in frase2)
```

```
'Python' no está en '¡Java es divertido!': True
'Python' no está en '¡Java es divertido, pero Python es realmente fantástico!':
False
```

Estos operadores de pertenencia son especialmente útiles al trabajar con cadenas y otros tipos de datos (como listas, conjuntos o diccionarios, que se cubrirán en capítulos posteriores).

3.8 Operador de exponenciación

Explica el concepto del operador de exponenciación en Python de manera lúdica.⁵⁴

¡Claro, hablemos del operador de exponenciación! Es como tener una varita mágica que puede hacer que los números crezcan súper rápido. Imagina que eres un mago y tienes una varita que puede multiplicar cualquier número por sí mismo tantas veces como quieras. Por ejemplo, si agitas tu varita y dices "2 elevado a la potencia de 3", obtienes $2 \times 2 \times 2 = 8$. Si la agitas de nuevo y dices "2 elevado a la potencia de 4", obtienes $2 \times 2 \times 2 \times 2 = 16$.

El operador de exponenciación en Python funciona de la misma manera: toma un número y lo eleva a una potencia (también conocida como exponente). Entonces, si quieres elevar 2 a la potencia de 3 en Python, simplemente puedes escribir `2**3`, lo que te dará el resultado 8. Puedes usar cualquier número como base y cualquier número entero como exponente, y Python calculará el resultado por ti. ¡Es como tener un poder mágico para hacer que los números crezcan!

Aquí tienes un ejemplo del uso del operador de exponenciación:

```
1 # Calculando el área de un cuadrado con longitud de lado 5
2 area = 5 ** 2
3 print(area) # Salida: 25
```

En este ejemplo, estamos utilizando el operador de exponente `**` para elevar la longitud del lado de un cuadrado a la potencia de 2, calculando así su área. El resultado se guarda en la variable `area`, que se imprime en la consola.

*¿Qué sucede cuando una expresión tiene múltiples operandos? ¿En qué orden se evalúan?*⁵⁵

Cuando una expresión tiene múltiples operandos, se evalúan según las reglas de precedencia de operadores y asociatividad. Los operadores con mayor precedencia se evalúan primero, seguidos por aquellos con menor precedencia.

Si hay operadores con la misma precedencia, su orden de evaluación depende de su asociatividad. Por ejemplo, los operadores aritméticos (+, -, *, /) tienen asociatividad de izquierda a derecha, lo que significa que los operandos (con igual precedencia) que se encuentren mas a la izquierda del inicio de la expresión se evalúan primero y se continúa evaluando hacia la derecha.

Aquí tienes un ejemplo para ilustrar esto. Considera la expresión `x = 2 + 3 * 4 / 2`. En esta expresión, los operadores de multiplicación y división tienen mayor precedencia que el operador de suma, e igual precedencia entre ellos. Por lo tanto, se evalúa `3 * 4` primero, lo que da como resultado 12. Luego, se evalúa `12 / 2`, lo que da como resultado 6. Finalmente, se evalúa `2 + 6`, lo que da como resultado 8. Por lo tanto, el valor de `x` sería 8.

Es importante tener en cuenta la jerarquía de la precedencia de los operadores y su asociatividad al escribir expresiones complejas en Python para asegurarse de que el código se comporte como se pretende. Esta jerarquía determina el orden en el que se evalúan los operadores en una expresión con múltiples operadores.

Para ayudarnos a recordar el orden de precedencia, podemos usar el acrónimo PEMDAS, que significa Paréntesis, Exponentes, Multiplicación y División, y Adición y Sustracción. Esto significa que los operadores dentro de paréntesis se evalúan primero, seguidos de los exponentes, luego la multiplicación y la división (que tienen la misma precedencia y se evalúan de izquierda a derecha), y finalmente la suma y la resta (que también tienen la misma precedencia y se evalúan de izquierda a derecha).

Sin embargo, es importante tener en cuenta que podemos cambiar el orden de evaluación utilizando paréntesis para agrupar ciertas operaciones juntas. Por ejemplo, en la expresión `(3 + 4) * 5`, la suma dentro de los paréntesis se evalúa primero, dando como resultado el valor de 7, que luego se multiplica por 5 para dar el resultado final de 35. En contraste, la expresión `3 + 4 * 5` da como resultado 23.

3.9 Operadores matemáticos avanzados

*¿Cómo puedo programar operaciones matemáticas más complejas como raíces cuadradas, logaritmos, y otros?*⁵⁶

Python incorpora un módulo pre-establecido llamado `math` que ofrece muchas funciones para realizar operaciones matemáticas complejas. Aquí tienes algunos ejemplos:

```

1 # Importar el módulo para operaciones matemáticas avanzadas
2 import math
3
4 # Calcular la raíz cuadrada de un número
5 num = 25
6 raiz_cuadrada_num = math.sqrt(num) # Utiliza la función sqrt() para calcular la raíz
   cuadrada
7 print(f"La raíz cuadrada de {num} es {raiz_cuadrada_num}")
8
9 # Calcular el logaritmo de un número en base 10
10 num2 = 1000
11 log_num2 = math.log10(num2) # Utiliza la función log10() para calcular el logaritmo en
   base 10
12 print(f"El logaritmo en base 10 de {num2} es {log_num2}")
13
14 # Obtener el valor de pi, una constante matemática
15 pi = math.pi
16 print(f"El valor de pi es {pi}")
17
18 # Convertir grados a radianes
19 grados = 45
20 radianes = math.radians(grados) # Utiliza la función radians() para convertir grados a
   radianes
21 print(f"{grados} grados equivale a {radianes} radianes")
22
23 # Calcular el seno de un ángulo en radianes
24 angulo = math.pi / 4
25 seno_angulo = math.sin(angulo) # Utiliza la función sin() para calcular el seno del á
   ngulo
26 print(f"El seno de {angulo} radianes es {seno_angulo}")

```

En este ejemplo, importamos el módulo `math` y utilizamos sus funciones integradas para realizar operaciones matemáticas avanzadas.

Calculamos la raíz cuadrada de un número utilizando la función `sqrt()`, el logaritmo en base 10 de un número utilizando la función `log10()`, y el seno de un ángulo utilizando la función `sin()`. También utilizamos la constante matemática predefinida `pi` y convertimos grados a radianes utilizando la función `radians()`.

Así, al utilizar el módulo `math` puedes incorporar fácilmente operaciones matemáticas avanzadas en tus programas de Python.

3.10 ¡Manos a la obra!

Sugiere algunos ejercicios divertidos e informativos para practicar los operadores aritméticos, de comparación, lógicos, a nivel de bits, de pertenencia, de exponenciación y matemáticos.⁵⁷

Aquí tienes algunos ejercicios de codificación para practicar varios operadores en Python:

1. Operadores aritméticos:

- Escribe un programa que calcule el área de un círculo con un radio dado, utilizando el operador `*` para la multiplicación y el operador `**` para la exponenciación.
- Escribe un programa que convierta una cantidad de minutos dada a horas y minutos utilizando los operadores `/` y `%`.
- Escribe un programa que convierta una hora en formato militar a formato civil.
- Escribe un programa que calcule el precio final, incluyendo el impuesto, dado el precio de un artículo y un porcentaje de IVA.

2. Operadores de comparación:

- Escribe un programa que verifique si un número dado es positivo, negativo o cero utilizando los operadores `>`, `<` y `==`.
- Escribe un programa que verifique si un número dado es par o impar utilizando el operador `%` para el módulo y el operador `==`.
- Escribe un programa que verifique si un año dado es bisiesto (es decir, divisible por 4 pero no divisible por 100, excepto los años divisible por 400) utilizando los operadores `%` y `AND`.

3. Operadores lógicos:

- Escribe un programa que, dada una lectura de temperatura, verifique si se encuentra dentro del rango normal de temperatura corporal humana.
- Escribe un programa que verifique si una frase dada contiene tanto las palabras "perro" como "gato" utilizando el operador `AND` y el operador `IN`.

4. Operadores a nivel de bits:

- Escribe un programa que cuente el número de bits establecidos en 1 en la representación binaria de un número entero dado en el rango `[0, 15]` (es decir, como máximo cuatro bits) utilizando los operadores `>>`, `&` y `+=`.
- Escribe un programa que invierta los bits de la representación binaria de un número entero dado en el rango `[0, 15]` (para mantener la representación binaria dentro de 4 bits), utilizando el operador `^`.
- Escribe un programa que, dado un número entero, imprima el doble de la cantidad sin utilizar operadores aritméticos.
- Escribe un programa que, dado un número entero par, imprima la mitad de la cantidad sin utilizar operadores aritméticos.

5. Operadores de pertenencia:

- Escribe un programa que verifique si un nombre dado comienza con una consonante (no con una vocal) utilizando los operadores NOT, OR e IN. Supón que el nombre comienza con una letra mayúscula.
- Escribe un programa que verifique si un correo electrónico ingresado por el usuario cumple con los criterios mínimos válidos, incluyendo la presencia del símbolo “arroba” (@) y el sufijo punto com.

6. Operador de exponenciación:

- Escribe un programa que calcule el cubo de un número dado utilizando el operador **.
- Escribe un programa que calcule la raíz cuadrada de un número dado utilizando el operador ** y un exponente fraccionario.

7. Operadores matemáticos:

- Escribe un programa que encuentre la hipotenusa de un triángulo rectángulo, dado las longitudes de los otros dos lados del triángulo.
- Escribe un programa que calcule la distancia entre dos puntos (x1, y1) y (x2, y2) utilizando el operador - para la resta, el operador ** para la exponenciación y la función matemática `sqrt()`.
- Escribe un programa que convierta grados a radianes y viceversa, utilizando las funciones matemáticas `radians()` y `degrees()`.
- Escribe un programa que calcule la altura de un objeto utilizando la longitud de su sombra y el ángulo de elevación del sol, utilizando la función trigonométrica `tan()`.

Ahora sugiere un problema de codificación interesante y atractivo para practicar la representación binaria de números decimales. Asume que tienes que mostrar una tabla de platos de comida, con columnas que representan los ingredientes utilizados para prepararlos y una columna final que muestra el código decimal del plato como el resultado de interpretar las columnas de ingredientes como bits de un número binario (muestra un ejemplo de una tabla con cuatro platos y cinco ingredientes). El programa debe solicitar el nombre de un plato y una serie de preguntas tipo sí/no para determinar el código decimal del plato, y luego imprimir una fila de la tabla con esta información (muestra un ejemplo de la salida en pantalla de la fila resultante).⁵⁸

Problema: Decodificador de Código de Platos

Se requiere una tabla de platos de comida con columnas que representan los ingredientes utilizados para prepararlos. Cada plato se identifica mediante un código decimal basado en sus ingredientes. Se te ha encargado construir un programa que solicite al usuario el nombre de un plato y una serie de preguntas tipo sí/no acerca de cada ingrediente, para determinar de esta manera el código que lo identifica.

A continuación una tabla de ejemplo con cuatro platos y cinco ingredientes:

Plato	Queso	Lechuga	Tomate	Cebolla	Carne	Código
Hamburguesa	No	Sí	Sí	Si	Sí	15
Pizza Margherita	Sí	No	Sí	No	No	20
Ensalada César	Sí	Sí	No	Sí	Sí	27
Perro Caliente	No	No	Si	Si	Si	7

Según las respuestas del usuario, el programa debe mostrar la fila correspondiente de la tabla, incluido el código decimal obtenido.

Ejemplo de salida en pantalla:

```

¡Bienvenido al Decodificador de Código de Platos!

Vamos a determinar el código del plato en función de los ingredientes.

Por favor, ingresa el nombre del plato: Hamburguesa

¿El plato contiene Queso? (S/N): N
¿El plato contiene Lechuga? (S/N): S
¿El plato contiene Tomate? (S/N): S
¿El plato contiene Cebolla? (S/N): S
¿El plato contiene Carne? (S/N): S

El código del plato es: 15

Aquí resumo el plato que seleccionaste:
| Plato | Queso | Lechuga | Tomate | Cebolla | Carne || Cód. |
|-----|-----|-----|-----|-----|-----||-----|
|Hamburguesa| Sí | Sí | Sí | No | Sí || 15 |

¡Gracias por usar el Decodificador de Código de Platos!
¡Buen provecho!

```

Puedes intentar agregar nuevos platos a la tabla, como pasta u omelette, y ver cómo el programa determina con precisión sus códigos numéricos basados en los ingredientes. Es una gran oportunidad para fortalecer tus habilidades en el manejo de la entrada del usuario y trabajar con representaciones binarias.

¡Disfruta construyendo tu Decodificador de Código de Platos!

(NB: Es posible que necesites utilizar la sentencia *if/else*, que se mencionó en el Capítulo 2, para procesar la entrada del usuario y calcular el código entero. Se cubrirá más sobre las declaraciones *if/else* en el Capítulo 4.)

La Torre de Control4

Enumera todas las estructuras de control en programación y proporciona una breve descripción de cada una.⁵⁹

En programación, existen varias estructuras de control que te permiten controlar el flujo de ejecución en tu código. Estas estructuras de control se pueden clasificar en tres tipos principales: estructuras de ramificación, estructuras de repetición y estructuras anidadas.

1. Estructuras de Ramificación (Condicionales):

- **Sentencia If/Else:** Ejecuta un bloque de código denotado `if`, si una cierta condición es verdadera. También puede incluir una declaración opcional `else` para proporcionar un bloque de código alternativo que se ejecutará si la condición es falsa.
- **Sentencia Match/case:** Evalúa una expresión y la compara con varios casos, ejecutando el bloque de código asociado con el caso coincidente.

2. Estructuras de Repetición (Bucles):

- **Bucle For:** Ejecuta un bloque de código un número específico de veces. Se utiliza con frecuencia cuando se conoce de antemano el número de iteraciones.
- **Bucle While:** Ejecuta un bloque de código repetidamente siempre que se cumpla una condición especificada. Es útil cuando el número de iteraciones no se conoce de antemano.
- **Bucle Do-While:** Similar al bucle **While**, pero garantiza que el bloque de código se ejecute al menos una vez antes de verificar el cumplimiento de la condición.

3. Estructuras Anidadas:

- **Condicionales Anidados:** Permite tener una sentencia `if/else` dentro de otra sentencia `if/else`. Proporciona una ramificación más compleja basada en múltiples condiciones.
- **Bucles Anidados:** Implica ejecutar un bucle dentro de otro bucle. Esto permite un control más intrincado sobre tareas repetitivas, con diferentes niveles de repetición.

Estas estructuras de control te brindan la capacidad de tomar decisiones, repetir acciones y concebir una lógica de programa más intrincada. Al combinar estas estructuras de manera efectiva, puedes crear programas potentes y flexibles para cumplir con diversos requisitos.

Así que veamos más de cerca cada una de ellas.

4.1 Sentencia condicional if/else

*Explicame todo lo que necesito saber sobre las sentencias if/else en Python de una manera divertida y creativa, con ejemplos didácticos.*⁶⁰

¡Claro! Imagina que eres un detective que resuelve casos en una ciudad llena de crimen. Tu trabajo consiste en tomar decisiones basadas en las pistas y pruebas que recopilas. Eso es precisamente de lo que tratan las instrucciones if/else: ¡tomar decisiones en tu código!

Aquí tienes un ejemplo didáctico: supongamos que eres un detective investigando un crimen misterioso. Tienes un sospechoso y necesitas determinar si es culpable o inocente. Puedes usar una instrucción if/else para ayudarte. La estructura general de esta instrucción condicional es:

```
1 if condicion:
2     # Código a ejecutar si la condición es verdadera
3 else:
4     # Código a ejecutar si la condición es falsa
```

Entonces, codifiquemos el ejemplo propuesto:

```
1 # Pide al usuario que ingrese su nombre
2 sospechoso = input("Ingresa tu nombre: ")
3
4 # Verifica si el nombre ingresado coincide con "Jack el Destripador"
5 if sospechoso == "Jack el Destripador":
6     # Si el nombre coincide, imprime un mensaje de arresto
7     print(f"¡Estás arrestado, {sospechoso}! ¡Eres culpable!")
8 else:
9     # Si el nombre no coincide, imprime un mensaje de inocencia
10    print(f"¡Puedes irte tranquilo, {sospechoso}! ¡Eres inocente!")
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/ykdth9bd>

Aquí tienes dos ejecuciones diferentes de este programa:

```
Ingresa tu nombre: Sherlock Holmes
¡Puedes irte tranquilo, Sherlock Holmes! ¡Eres inocente!
```

```
Ingresa tu nombre: Jack el Destripador
¡Estás arrestado, Jack el Destripador! ¡Eres culpable!
```

Además, también puedes usar instrucciones if/elif/else para manejar más de dos escenarios diferentes. Estos escenarios pueden definirse combinando varias condiciones mediante

operadores lógicos. Supongamos que quieres determinar si es un buen día para ir a la playa según el clima. Aquí tienes un ejemplo:

```
1 # Establece las condiciones iniciales
2 soleado = True
3 temperatura = 22
4
5 # Si está soleado y la temperatura es superior a 25 grados
6 if soleado and temperatura > 25:
7     print("¡Es un día perfecto para ir a la playa!")
8 # Si está soleado pero la temperatura es de 25 grados o inferior
9 elif soleado and temperatura <= 25:
10     print("Hace un poco de frío, ¡pero aún puedes disfrutar de la playa!")
11 # Si no está soleado
12 else:
13     print("Tal vez sea mejor quedarse en casa hoy.")
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/362yh7w6>

En este código, tenemos dos variables: `soleado` y `temperatura`. Analicemos el código y comprendamos cómo se evalúan las condiciones y el papel de cada variable:

1. Se definen las condiciones iniciales `soleado=True` y `temperatura=22`.
2. La primera instrucción condicional utiliza la expresión lógica `if soleado and temperatura > 25`. Esta condición verifica si está soleado (`soleado == True`) y si la temperatura es superior a 25 grados (`temperatura > 25`). Si ambas condiciones son verdaderas, se ejecutará el código dentro de este bloque y se imprimirá "¡Es un día perfecto para ir a la playa!".
3. Si no se cumple la primera condición, el programa pasa a la siguiente instrucción condicional, `elif soleado and temperatura <= 25`. Esta condición verifica si está soleado (`soleado == True`) y si la temperatura es de 25 grados o inferior (`temperatura <= 25`). Si ambas condiciones son verdaderas, se ejecutará el código dentro de este bloque y se imprimirá "Hace un poco de frío, ¡pero aún puedes disfrutar de la playa!".
4. Si no se cumplen ni la primera ni la segunda condición, el programa pasa al bloque `else`. Este bloque se ejecutará solo cuando no esté soleado (`soleado == False`). En este caso, imprimirá "Tal vez sea mejor quedarse en casa hoy."

A continuación la salida en pantalla del código con las condiciones iniciales mencionadas:

```
Hace un poco de frío, ¡pero aún puedes disfrutar de la playa!
```

El orden de las condiciones es importante porque el algoritmo las evalúa una por una de arriba hacia abajo. Una vez se cumpla una condición, se ejecuta el bloque de código correspondiente y el programa sale de la estructura `if/elif/else` para continuar con el resto del código. Esto significa que solo se ejecutará un bloque de código, según qué condición sea verdadera primero. Observa que en este caso, la variable `temperatura` no afecta la ejecución del bloque `else` porque el flujo del algoritmo solo llega a este punto cuando las condiciones `if` y `elif` anteriores que involucran la variable `soleado` no se cumplen, es decir, cuando `soleado == False`. Por lo tanto, el bloque `else` se ejecutará cuando no esté soleado, independientemente del valor de la variable `temperatura`.

Recuerda, las instrucciones `if/else` pueden manejar varias condiciones y permiten que tu programa tome diferentes decisiones basadas en esas condiciones. Son como las elecciones que haces en un juego, que finalmente determinan el resultado.

Entonces, ya sea que estés resolviendo misterios, planificando viajes a la playa o creando programas emocionantes, las instrucciones `if/else` en Python son tus herramientas confiables para tomar decisiones en tus algoritmos.

*¿Hay algo más que deba saber sobre las estructuras condicionales en Python?*⁶¹

Sí, hay algunas cosas más que debes saber sobre las estructuras condicionales en Python:

- **Instrucciones condicionales anidadas:** Puedes tener instrucciones condicionales dentro de otras instrucciones condicionales para manejar escenarios más complejos. Esto se conoce como instrucciones condicionales anidadas. Dentro de un bloque `if` o `elif`, puedes tener otra instrucción `if` para refinar aún más las condiciones con criterios adicionales.

(NB: Cubriremos más sobre las estructuras anidadas al final del capítulo.)

- **Instrucciones condicionales encadenadas:** En algunos casos, puedes tener múltiples condiciones que se superponen, lo que significa que el algoritmo puede ejecutar varios bloques `if` según las condiciones que apliquen al valor de la variable. La sintaxis para las instrucciones condicionales encadenadas es la siguiente:

```
1 if condicion1:
2     # bloque de código
3 if condicion2:
4     # bloque de código
5 if condicion3:
6     # bloque de código
```

En este caso, cada instrucción `if` se evaluará de forma independiente y sus bloques de código respectivos se ejecutarán cuando sus condiciones sean `True`.

Aquí tienes un ejemplo de instrucciones condicionales encadenadas donde las condiciones pueden superponerse, lo que resulta en la ejecución de varios bloques `if`:

```

1 # Solicita al usuario su edad
2 edad = int(input("¿Cuántos años tienes? "))
3
4 # Verifica etapas de la edad usando condicionales encadenados
5 if edad > 13:
6     print("¡Felicidades, has sobrevivido a tu infancia!")
7 if edad > 18:
8     print("¡Has superado la montaña rusa de los años adolescentes!")
9 if edad > 35:
10    print("¡Ahora eres un adulto de verdad, y las facturas son tus mejores amigos!"
11    )
12 if edad >= 65:
13    print("¡Bienvenido al exclusivo club de los ciudadanos mayores! ¡Descuentos te
    esperan!")

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2s42xwnc>

En este código, las condiciones no son excluyentes, y cada bloque se ejecutará si la edad cumple con la condición correspondiente. Por ejemplo, si el usuario ingresa una edad de 15, el programa imprimirá solamente el mensaje "¡Felicidades, has sobrevivido a tu infancia!".

Ahora, si el usuario ingresa una edad de 25, el programa imprimirá tanto "¡Felicidades, has sobrevivido a tu infancia!" como "¡Has superado la montaña rusa de los años adolescentes!". Veamos qué sucede si el usuario tiene una edad de 40:

```

¿Cuántos años tienes? 40
¡Felicidades, has sobrevivido a tu infancia!
¡Has superado la montaña rusa de los años adolescentes!
¡Ahora eres un adulto de verdad, y las facturas son tus mejores amigos!

```

Como puedes ver, estas salidas reflejan las etapas que el usuario ha superado según su edad, lo que permite que varias condiciones sean verdaderas simultáneamente.

- **Evaluación de corto circuito:** Python utiliza la evaluación de corto circuito en las instrucciones condicionales con condición compuesta. Esto significa que si el resultado de una condición compuesta se puede determinar en función de la primera parte de la condición, la segunda parte no se evaluará. Por ejemplo, en una condición `and`, si el primer operando es `False`, el segundo operando no se evaluará, ya que por definición, en ese caso el resultado total del `and` siempre será `False`.
- **Operador ternario:** Python también proporciona una forma compacta de escribir expresiones condicionales simples llamado operador ternario. Te permite evaluar una condición y devolver un valor si la condición es verdadera y otro valor si la condición es falsa. La sintaxis es la siguiente:

```
1 valor = valor_verdadero if condicion else valor_falso
```

Por ejemplo:

```
1 # Solicita al usuario la velocidad de su vehículo
2 velocidad = int(input("Oficial: ¿Cuál era la velocidad de su vehículo en mph? "))
3
4 # Usa el operador ternario para definir mensaje de respuesta
5 respuesta = "multa por exceso de velocidad, lo siento" if velocidad > 70 else "¡
    choca esos cinco!"
6 print(f"Oficial: Muy bien, es posible que tenga que hacerle un ... {respuesta}. ¡
    Conduzca con seguridad!")
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/e6b9v554>

En este código, la instrucción `print()` incorpora en la respuesta del oficial el resultado del operador ternario, lo que da como resultado un mensaje con un toque de humor emitiendo una "multa por exceso de velocidad" si la velocidad está por encima de 70 mph, y un divertido "¡choca esos cinco!" si la velocidad está dentro del límite. El mensaje también fomenta una conducción segura. Veamos:

```
Oficial: ¿Cuál era la velocidad de su vehículo en mph? 45
Oficial: Muy bien, es posible que tenga que hacerle un... ¡choca esos cinco!.
    ¡Conduzca con seguridad!
```

```
Oficial: ¿Cuál era la velocidad de su vehículo en mph? 85
Oficial: Muy bien, es posible que tenga que hacerle un... multa por exceso de
    velocidad, lo siento. ¡Conduzca con seguridad!
```

Así pues, los aspectos mencionados anteriormente son consideraciones adicionales a tener en cuenta al trabajar con estructuras condicionales en Python. Tener una comprensión clara de estos conceptos te permitirá escribir código más flexible y eficiente cuando te enfrentes a escenarios que involucren toma de decisiones complejas.

4.2 Sentencia `match/case`

*Ahora hablemos sobre las sentencias `match/case` en Python. Explícame todo lo que necesito saber sobre ellas de una manera creativa, utilizando ejemplos divertidos.*⁶²

A partir de Python 3.10, se introduce la sentencia `match/case` como una nueva característica llamada “coincidencia de patrón estructurada”. Proporciona una forma poderosa y concisa de hacer coincidir, extraer valores y decidir casos o ramas basados en patrones.

La sintaxis básica de la sentencia `match/case` en Python es la siguiente:

```

1 match expresion:
2     case valor1:
3         # Código a ejecutar si la expresión coincide con valor1
4     case valor2:
5         # Código a ejecutar si la expresión coincide con valor2
6     ...
7     case valorN:
8         # Código a ejecutar si la expresión coincide con valorN
9     case _:
10        # Código a ejecutar si no hay coincidencias anteriores

```

Aquí tienes un desglose de la sintaxis:

- **expresión:** La expresión cuyo valor se comparará con los casos.
- **valor1, valor2, ..., valorN:** Los posibles valores con los que la expresión puede coincidir. Estos pueden ser constantes, variables o incluso patrones.
- **_:** El caso comodín, que coincide con cualquier valor que no se haya enumerado explícitamente en los casos anteriores.

El código dentro de cada bloque de caso se ejecuta cuando la expresión coincide con el valor correspondiente. Si varios casos coinciden, solo se ejecuta el código del primer caso coincidente.

La sentencia `match/case` proporciona una forma concisa de manejar múltiples casos según el valor de una expresión, lo que hace que el código sea más legible y elimina la necesidad de largas cadenas de sentencias `if/elif/else`. Recuerda que la sentencia `match/case` se introdujo en Python 3.10, así que asegúrate de tener instalada la versión adecuada para utilizar esta característica.

Exploremos la sentencia `match/case` utilizando un ejemplo curioso de pedido de pizza. Imagina que eres un amante de la pizza y quieres pedir una pizza con diferentes ingredientes. Veamos cómo podemos usar la sentencia `match/case` para manejar esta situación:

```

1 pizza = input("¿Qué pizza te gustaría hoy? (vegetariana/carnívora/hawaiana)")
2
3 match pizza.lower():
4     case "vegetariana":
5         ingredientes = "Champiñones, aceitunas y pimientos"
6     case "carnívora":
7         ingredientes = "Carne molida, salchicha y pepperoni"
8     case "hawaiana":
9         ingredientes = "Jamón, piña y queso extra"
10    case _:
11        ingredientes = "Tomate y queso solamente"

```

```

12
13 print(f"Has pedido una pizza {pizza} con los ingredientes: {ingredientes}")

```

En este ejemplo, primero le preguntamos al usuario qué tipo de pizza quiere pedir. Los ingredientes correspondientes se determinan utilizando la sentencia `match/case`. Cada `case` en el bloque `match` verifica si el nombre de la pizza coincide con uno de los valores especificados. Si se encuentra una coincidencia, los ingredientes correspondientes se asignan a la variable `ingredientes`. Si ninguno de los casos coincide con la pizza, se activa el caso comodín `"_"`, indicando una pizza básica:

```

¿Qué pizza te gustaría hoy? (vegetariana/carnívora/hawaiana) Hawaiana
Has pedido una pizza Hawaiana con los ingredientes: Jamón, piña y queso extra

```

```

¿Qué pizza te gustaría hoy? (vegetariana/carnívora/hawaiana) isorpresa!
Has pedido una pizza isorpresa! con los ingredientes: Tomate y queso solamente

```

Con la sentencia `match/case`, puedes manejar elegantemente diferentes casos según el valor que se esté comparando. Mejora la legibilidad y reduce la necesidad de estructuras complejas `if/elif/else`.

De acuerdo. En la sentencia `match`, veo que usaste el comando `"lower()"`. ¿Debo usarlo cada vez que codifique una sentencia `match/case`?⁶³

No, el uso del comando `lower()` en la sentencia `match` fue específico del ejemplo de las pizzas. Se utilizó para convertir la entrada del usuario a minúsculas y así realizar una coincidencia que no distinga entre mayúsculas y minúsculas. Esto significa que, independientemente de si el usuario ingresa "vegetariana", "VEGETARIANA" o "VeGetAriana", se considerará una coincidencia.

En general, no es necesario usar `lower()` en cada sentencia `match/case`. Solo es necesario cuando deseas realizar coincidencias sin tener en cuenta mayúsculas y minúsculas. Si deseas realizar una coincidencia sensible a las mayúsculas y minúsculas, puedes omitir el comando `lower()`.

4.3 Sentencia de bucle `for`

Por favor, proporciona una breve definición técnica del bucle `for`.⁶⁴

¡Por supuesto! Un bucle `for` es una estructura de programación que nos permite ejecutar repetidamente un bloque de código un número específico de veces o para cada elemento en una colección. Su sintaxis general en Python es la siguiente:


```

1 for variable in iterable:
2     # Bloque de código a ejecutar

```

En esta sintaxis:

- **variable:** Es una variable que toma el valor de cada elemento en el **iterable** durante cada iteración (repetición) del bucle.
- **iterable:** Representa una secuencia o colección de elementos, como un rango, una lista o una cadena.
- **bloque de código:** Es el bloque de código indentado bajo la sentencia **for**, que consiste del conjunto de instrucciones que se ejecutarán repetidamente.

Durante cada iteración del bucle, la **variable** toma el valor del siguiente elemento en el **iterable**, y enseguida se ejecuta el bloque de código. Este proceso continúa hasta que se han procesado todos los elementos en el **iterable**.

El bucle **for** es especialmente útil cuando conocemos de antemano el número de iteraciones que queremos que el bloque de código se repita o cuando queremos iterar sobre una colección de elementos. Simplifica tareas repetitivas y nos permite realizar operaciones de manera eficiente sin escribir código redundante.

Ahora veamos los bucles for en acción de manera divertida y creativa, utilizando ejemplos lúdicos.⁶⁵

¡Por supuesto! Exploraremos el mundo de las sentencias de bucle **for** en Python con algunos ejemplos lúdicos.

1. *Contando con facilidad.* El bucle **for** te permite contar números sin esfuerzo dentro de un rango específico. Por ejemplo, para contar del 1 al 10, puedes usar la función **range()**:

```

1 # range(1, 11) genera una secuencia del 1 al 10 (es decir a 11-1)
2 for numero in range(1, 11):
3     print(numero)

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/pcvbud98>

Este código imprimirá iterativamente los números del 1 al 10, facilitando el conteo.

```

1
2
3
4
5
6
7

```

```
8
9
10
```

2. *Avanzando hacia el éxito.* También puedes controlar el tamaño del paso al iterar sobre un rango. Supongamos que deseas contar del 1 al 10, pero solo incluir los números pares. Con el parámetro de paso, puedes lograr esto:

```
1 # el último parámetro en range() es el tamaño del paso
2 for numero in range(2, 11, 2):
3     print(numero)
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/ycysvpds>

En este caso, el bucle comienza desde 2 y avanza de 2 en 2 hasta llegar a 10, imprimiendo solo los números pares.

```
2
4
6
8
10
```

3. *Desvelando la palabra mágica.* El bucle `for` no se limita a los números; también puede iterar sobre los caracteres de una cadena. Digamos que tenemos una palabra mágica, “abracadabra”, y queremos revelar sus letras una por una:

```
1 palabra_magica = "abracadabra"
2 print(f"Vamos a desvelar las letras mágicas de '{palabra_magica}': ")
3 for letra in palabra_magica:
4     print("+---+")
5     print(f"| {letra} |")
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2p8vdpnm>

Al ejecutar este código, se mostrará cada letra de la palabra mágica dentro de cajas mágicas en líneas separadas, desvelando su poder oculto.

```
Vamos a desvelar las letras mágicas de 'abracadabra':
+---+
| a |
+---+
| b |
+---+
| r |
```

```

+---+
| a |
+---+
| c |
+---+
| a |
+---+
| d |
+---+
| a |
+---+
| b |
+---+
| r |
+---+
| a |

```

4. *Hechizo de multiplicación.* El bucle **for** se puede utilizar para generar tablas de multiplicación sin esfuerzo. Supongamos que deseas imprimir la tabla de multiplicar de un número dado, del 1 al 10:

```

1 numero = int(input("Ingresa un número para lanzar un hechizo de multiplicación: "))
2 for i in range(1, 11):
3     resultado = numero * i
4     print(f"{numero} x {i} = {resultado}")

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/3fmfxdap>

Al ejecutar este código, se mostrará la tabla de multiplicar para el número ingresado por el usuario, mostrando cada operación de multiplicación junto con el resultado correspondiente.

```

Ingresa un número para arrojar un hechizo de multiplicación: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

```

5. *Transformación de cadenas.* El bucle `for` también se puede usar para manipular cadenas, como convertir entre mayúsculas y minúsculas cada carácter. Digamos que tenemos una cadena, "¡Hola, Mundo!", y queremos convertirla a mayúsculas:

```

1 texto = "¡Hola, Mundo!"
2 texto_convertido = ""
3 for caracter in texto:
4     texto_convertido += caracter.upper()
5 print("Texto original: ", texto)
6 print("Texto convertido: ", texto_convertido)

```

Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2p8kw9ee>

Al ejecutar este código, se imprimirá "¡HOLA, MUNDO!", transformando todos los caracteres de la cadena a mayúsculas.

```

Texto original:  ¡Hola, Mundo!
Texto convertido: ¡HOLA, MUNDO!

```

Como puedes ver, el bucle `for` es una estructura poderosa que simplifica la iteración y te permite realizar una amplia variedad de tareas, desde generar patrones matemáticos hasta manipular cadenas. Su versatilidad y facilidad de uso lo convierten en una herramienta valiosa en tu camino como programador.

4.4 Sentencia de bucle `while`

*Proporciona una breve definición técnica del bucle `while`.*⁶⁶

El bucle `while` es una sentencia de flujo de control en la programación que te permite ejecutar repetidamente un bloque de código siempre que se cumpla una condición especificada. Proporciona una forma de crear bucles que iteran dinámicamente en función de la evaluación de una condición, en lugar de un número predeterminado de iteraciones. Este tipo de bucle consta de una condición que se verifica antes de cada iteración, y mientras la condición siga siendo `True`, se ejecuta el bloque de código dentro del bucle. El bucle continúa hasta que la condición se convierte en `False`, momento en el que el programa sale del bucle y continúa con el siguiente conjunto de instrucciones.

La sintaxis del bucle `while` es la siguiente:

```

1 while condicion:
2     # Bloque de código a ejecutar

```

En esta sintaxis, la `condicion` es una expresión booleana que se evalúa antes de cada iteración. Mientras la `condicion` sea verdadera, se ejecutará el **bloque de código**; si es falsa, se sale del

bucle. Dentro del **bloque de código**, puedes realizar cualquier operación o cálculo deseado. Después de que se ejecute el **bloque de código**, el programa regresa al comienzo del bucle y vuelve a evaluar la condición para determinar si continuar o salir del bucle.

Es importante asegurarse de que la **condición** eventualmente se vuelva falsa para evitar bucles infinitos, que son bucles que continúan ejecutándose indefinidamente sin detenerse, lo que provocará que el programa se bloquee. Para evitar esto, es crucial diseñar la **condición** de manera que permita la terminación. Esto se puede lograr incorporando variables o instrucciones dentro del bucle que actualicen o modifiquen la **condición**, asegurándose de que eventualmente se evalúe como falsa y se salga del bucle.

Ahora veamos los bucles while en acción de una manera divertida y creativa, utilizando ejemplos lúdicos.⁶⁷

¡Por supuesto! Exploraremos los bucles **while** a través de algunos ejemplos lúdicos:

1. *Contando ovejas.* Imagina que estás tratando de quedarte dormido, pero no puedes conciliar el sueño. Escribamos un programa que cuente ovejas hasta que finalmente te duermas. Aquí está el código:

```

1 # Inicializar la variable ovejas en 0
2 ovejas = 0
3
4 # Solicitar al usuario su estado de sueño
5 dormido = input("¿Ya estás dormido (si/no)? ")
6
7 # Continuar el bucle mientras el usuario no esté dormido
8 while dormido.lower() != "si":
9     ovejas += 1
10    print(f"Contando oveja #{ovejas}: Baaa...\n")
11    dormido = input("¿Ya estás dormido (si/no)? ")
12
13 # Una vez que se termina el bucle, indicar que el usuario está dormido
14 print("¡Te has dormido! ¡Dulces sueños!")

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/4mp2ty8n>

En este código, simulamos contar ovejas para ayudar al usuario a quedarse dormido. El programa solicita al usuario su estado de sueño utilizando la función `input()`. El bucle **while** continúa mientras la respuesta del usuario no sea "si". Dentro del bucle, incrementamos la cantidad de **ovejas** en 1 e imprimimos un mensaje para ayudar al usuario a visualizar una oveja saltando sobre una cerca. Luego, solicitamos nuevamente al usuario su estado de sueño. Una vez que el usuario responde con "si", la condición del bucle **while** se vuelve **False**, el programa termina el bucle e imprime un mensaje indicando que el usuario se ha quedado dormido.

Es importante tener en cuenta que en este ejemplo no sabemos de antemano cuántas veces se repetirá el bucle. El bucle continúa hasta que el usuario responda con “sí” y la cantidad de veces que se repite el bucle depende de la entrada del usuario.

```
¿Ya estás dormido (sí/no)? no
Contando ovejas #1: Baaa...

¿Ya estás dormido (sí/no)? no
Contando ovejas #2: Baaa...

¿Ya estás dormido (sí/no)? no
Contando ovejas #3: Baaa...

¿Ya estás dormido (sí/no)? sí
¡Te has dormido! ¡Dulces sueños!
```

2. *Abriendo el cofre del tesoro.* Imagina que eres un cazador de tesoros que intenta abrir un cofre misterioso. Desafortunadamente, el cofre está cerrado con llave, y necesitas seguir adivinando hasta encontrar el código secreto que lo abre. Escribamos un programa que simule esta búsqueda del tesoro:

```
1 # Definir una palabra secreta y generar el código secreto
2 palabra_secreta = "AbreteSesamo"
3 codigo_secreto = palabra_secreta.swapcase()
4
5 # Solicitar al usuario que ingrese su intento
6 intento = input("Ingresa el código secreto: ")
7
8 # Continuar el bucle mientras la suposición sea incorrecta
9 while intento != codigo_secreto:
10     print("¡Código incorrecto! Inténtalo nuevamente.\n")
11     intento = input("Ingresa el código secreto: ")
12
13 # Imprimir un mensaje indicando que el cofre está abierto
14 print("¡Felicidades! ¡Has desbloqueado el cofre del tesoro!")
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/23dudzsz>

En este código, simulamos un código secreto que debe adivinarse correctamente para desbloquear un cofre del tesoro. La `palabra_secreta` está definida como “AbreteSesamo”. Generamos el `codigo_secreto` intercambiando las mayúsculas y minúsculas de los caracteres en `palabra_secreta` utilizando el método `swapcase()`.

El programa solicita al usuario que ingrese su intento utilizando la función `input()`. El bucle `while` continúa mientras el intento no sea igual al `codigo_secreto`. Dentro del bucle, imprimimos un mensaje indicando que el `intento` es incorrecto y solicitamos nuevamente al usuario que ingrese uno nuevo. Una vez que el usuario ingresa el `codigo_secreto` correcto, la condición del bucle se convierte en `False`, y el programa sale del bucle. Finalmente, imprimimos un mensaje felicitando al usuario por desbloquear el cofre del tesoro.

```
Ingresa el código secreto: AbreteSesamo
¡Código incorrecto! Inténtalo nuevamente.

Ingresa el código secreto: abretesesamo
¡Código incorrecto! Inténtalo nuevamente.

Ingresa el código secreto: aBRETESesAMO
¡Felicidades! ¡Has desbloqueado el cofre del tesoro!
```

Estos ejemplos lúdicos demuestran cómo se pueden usar bucles `while` para crear programas interactivos y atractivos. Al utilizar escenarios creativos, podemos hacer que la programación sea más divertida y agradable.

*En el ejemplo “Contando ovejas”, preguntas al usuario si está dormido dos veces. ¿No es redundante? ¿Hay alguna forma de escribir el bucle sin hacer la misma pregunta dos veces?*⁶⁸

Tienes razón, hacer la misma pregunta al usuario dos veces en el ejemplo “Contando ovejas” es redundante. Podemos simplificar el bucle y evitar repetir la misma pregunta dos veces utilizando un bucle `while True` con una sentencia condicional que lo interrumpa cuando el usuario responda “sí”, usando para ello el comando `break`. Aquí tienes una versión actualizada del código que ilustra esta alternativa:

```
1 ovejas = 0
2
3 while True:
4     dormido = input("¿Ya estás dormido (sí/no)? ")
5     if dormido == "sí":
6         break
7     ovejas += 1
8     print(f"Contando oveja #{ovejas}: ¡Baaa...!")
9
10 print("¡Te has dormido! ¡Dulces sueños!")
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/3vrcnn2f>

En esta versión alternativa, el bucle `while True` continuará indefinidamente hasta que se encuentre la sentencia `break`. Dentro del bucle, solicitamos al usuario su estado de sueño utilizando la función `input()`. Si el usuario responde con “sí”, salimos del bucle e imprimimos el mensaje de despedida. De lo contrario, incrementamos el contador de `ovejas` en 1 e imprimimos un mensaje para visualizar una oveja saltando sobre una cerca. Al utilizar este enfoque, logramos que la pregunta al usuario aparezca una sola vez dentro del código.

*¿Hay algo más que deba saber sobre los bucles `for` o `while` en Python?*⁶⁹

Aquí tienes algunos puntos adicionales que debes tener en cuenta sobre los bucles `for` y `while` en Python:

► Bucles `for`:

- La función `range()` se usa comúnmente en bucles `for` para generar una secuencia de números que se puede iterar.
- La sentencia `break` se puede utilizar para salir prematuramente de un bucle `for` antes de completar todas las iteraciones.
- La sentencia `continue` se puede utilizar para saltar la iteración actual y continuar con la siguiente iteración del bucle.
- Puedes recorrer varias estructuras de datos en Python, incluyendo cadenas, listas, tuplas y diccionarios.

(NB: Más sobre estas estructuras en el próximo capítulo)

► Bucles `while`:

- La condición en un bucle `while` se evalúa antes de cada iteración, y el bucle continúa mientras la condición siga siendo verdadera.
- Es importante asegurarse de que la condición eventualmente se vuelva falsa para evitar bucles infinitos, que pueden hacer que el programa se ejecute indefinidamente.
- La sentencia `break` se puede utilizar para salir prematuramente de un bucle `while` basado en cierta condición.
- La sentencia `continue` se puede utilizar para saltar la iteración actual y continuar con la siguiente iteración del bucle.
- Los bucles `while` son útiles cuando no se sabe de antemano cuántas veces debe iterar el bucle, ya que el número de iteraciones depende de la evaluación de la condición.

Comprender cómo usar los bucles de manera efectiva puede mejorar enormemente tu capacidad para iterar sobre datos, realizar tareas repetitivas y resolver problemas complejos en Python. Siempre es una buena práctica probar y depurar tus bucles para asegurarte de que se comporten como se espera y manejen diferentes escenarios correctamente.

4.5 Estructuras de control anidadas

*¿Qué es una estructura de control anidada y por qué es útil?*⁷⁰

Una estructura de control anidada es una estructura de control (por ejemplo, un condicional `if`, un bucle `for` o un bucle `while`) que se encuentra dentro de otra estructura de control. Permite tener múltiples niveles de ramificación o bucles dentro de un programa.

Las estructuras de control anidadas son herramientas poderosas en la programación que permiten manejar lógica compleja y realizar tareas avanzadas. Facilitan la creación de procesos de toma de decisiones intrincados y operaciones iterativas, proporcionando flexibilidad y adaptabilidad a los programas. Con estructuras de control anidadas, puedes diseñar código que responda a múltiples condiciones, realice tareas repetitivas con complejidad variable e implemente algoritmos sofisticados. Este enfoque dinámico hace que los programas sean más eficientes y capaces de resolver problemas complejos, ya que pueden adaptarse a condiciones cambiantes, tomar decisiones basadas en múltiples factores y ejecutar operaciones iterativas a diferentes niveles de profundidad.

Sin embargo, es importante utilizar las estructuras de control anidadas con prudencia y mantener la legibilidad del código. A medida que aumenta la profundidad del anidamiento, el código puede volverse más difícil de entender y depurar. Por lo tanto, es crucial encontrar un equilibrio entre utilizar estructuras de control anidadas para abordar escenarios complejos y mantener el código fácil de mantener y comprender.

*Ahora adentrémonos en las estructuras de control anidadas en Python. Muéstrame cómo usar condicionales anidados permitiendo múltiples niveles de ramificación, bucles anidados incluyendo bucles `for` dobles, bucles `while` dobles y bucles mixtos `while/for` o `for/while`, así como combinaciones de bucles dentro de condicionales y condicionales dentro de bucles. Utiliza ejemplos de superhéroes y cómics para ser creativo y lúdico en tu explicación.*⁷¹

¡Claro! Sumerjámonos en el emocionante mundo de los superhéroes para explorar el concepto de estructuras de control anidadas en Python.

1. *Superhéroe vs Supervillano.* Aquí tienes un programa de ejemplo que pide al usuario que elija un superhéroe y un supervillano, y luego determina quién vence a quién utilizando condicionales anidados de dos niveles.

En este ejemplo, los condicionales anidados te permiten manejar diversas combinaciones de elecciones de superhéroes y supervillanos. Cada bloque anidado verifica el superhéroe y supervillano seleccionados y, en función de la combinación, muestra un mensaje correspondiente de “quién vence a quién”. Si la combinación elegida no coincide con ninguno de los condicionales anidados, muestra un mensaje de error apropiado.

Demos un vistazo al código:

```

1 # Obtener las entradas del usuario
2 superhero = input("Ingresa un superhéroe: ")
3 supervillain = input("Ingresa un supervillano: ")
4
5 # Verificar la combinación de superhéroe/supervillano
6 if superhero == "Superman":
7     if supervillain == "Lex Luthor":
8         print("¡Superman vence a Lex Luthor!")
9     elif supervillain == "Doomsday":
10        print("¡Doomsday vence a Superman!")
11    else:
12        print("¡Elección de supervillano no válida!")
13 elif superhero == "Batman":
14     if supervillain == "Joker":
15         print("¡Batman vence al Joker!")
16     elif supervillain == "Bane":
17         print("¡Bane vence a Batman!")
18     else:
19         print("¡Elección de supervillano no válida!")
20 elif superhero == "Spider-Man":
21     if supervillain == "Green Goblin":
22         print("¡Spider-Man vence al Green Goblin!")
23     elif supervillain == "Venom":
24         print("¡Venom vence a Spider-Man!")
25     else:
26         print("¡Elección de supervillano no válida!")
27 else:
28     print("¡Elección de superhéroe no válida!")

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/yesc8st4>

Aquí tienes algunos ejemplos de salida en pantalla de este programa:

```

Elige un superhéroe: Batman
Elige un supervillano: Joker
¡Batman vence al Joker!

```

```

Elige un superhéroe: Spider-Man
Elige un supervillano: Venom
¡Venom vence a Spider-Man!

```

2. *Sesiones de Entrenamiento de Superhéroes*. Ahora veamos bucles anidados `for` utilizando un escenario de sesiones de entrenamiento de superhéroes:

```

1 import time
2 num_sesiones = int(input("Ingresa el número de sesiones de entrenamiento: "))
3 duracion_sesion = int(input("Ingresa la duración de cada sesión de entrenamiento (
    en segundos): "))
4
5 for sesion in range(1, num_sesiones + 1):
6     print(f"\nSesión de entrenamiento {sesion}:")
7     for segundo in range(1, duracion_sesion + 1):
8         print(f"Segundos transcurridos: {segundo}. Entrenamiento en progreso...")
9         time.sleep(1) # Pausa de un segundo
10    print("Sesión completa. Descansa 5 segundos y prepárate para la próxima sesión
    ...")
11    time.sleep(5) # Pausa de cinco segundos
12
13 print("\n¡Bien hecho! ¡Nos vemos pronto para más entrenamiento!")

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/5yhkw5wd>

En este ejemplo, tenemos bucles anidados para simular múltiples sesiones de entrenamiento con una duración específica. El bucle externo itera sobre las sesiones de entrenamiento, mientras que el bucle interno itera sobre la duración de cada sesión. Imprime el número de sesión y muestra el progreso luego de que pase cada segundo. Después de completar la sesión, muestra un mensaje que indica que la sesión está completa y luego repite el proceso para la siguiente sesión.

Observa que importamos el módulo `time`, que proporciona la función `sleep()`, que se utiliza para pausar la ejecución del programa durante el número especificado de segundos. Lo utilizamos dentro del bucle interno para crear una pausa de un segundo entre cada minuto de entrenamiento. Aquí tienes un ejemplo de la salida de este programa:

```

Ingresa el número de sesiones de entrenamiento: 2
Ingresa la duración de cada sesión de entrenamiento (en segundos): 2

Sesión de entrenamiento 1:
Segundos transcurridos: 1. Entrenamiento en progreso...
Segundos transcurridos: 2. Entrenamiento en progreso...
Sesión completa. Descansa 5 segundos y prepárate para la próxima sesión...

Sesión de entrenamiento 2:
Segundos transcurridos: 1. Entrenamiento en progreso...

```

Segundos transcurridos: 2. Entrenamiento en progreso...

Sesión completa. Descansa 5 segundos y prepárate para la próxima sesión...

¡Bien hecho! ¡Nos vemos pronto para más entrenamiento!

3. *¡Vivas a los Superhéroes!:* Puedes utilizar bucles **while** anidados para repetir un conjunto de acciones para una secuencia de superhéroes. Veamos un ejemplo donde vitoreamos a diferentes superhéroes:

```
1 import random
2 # El bucle externo se ejecuta hasta que se ingrese "salir"
3 while True:
4     superheroe = input("\nIngresa el nombre de un superhéroe (o 'salir' para
5     terminar): ")
6     if superheroe == "salir":
7         break;
8     # Generamos un número aleatorio de vítores utilizando randint()
9     vivas = random.randint(1, 10)
10    # El bucle interno se ejecuta mientras haya vítores restantes
11    while vivas > 0:
12        print(f"¡Viva {superheroe}!") # Imprimimos el mensaje de vitoreo
13        vivas -= 1 # Disminuimos la cantidad de vítores
14    print("\n¡Adiós!")
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/yc6pryxh>

En este código, se utiliza la función `random.randint()` para generar un número aleatorio de ¡vivas! para cada superhéroe. El bucle `while` externo permite al usuario ingresar nombres de superhéroes hasta que se escriba “salir”, mientras que el bucle `while` interno controla la cantidad de ¡vivas! que se imprimirán para cada superhéroe. Al ejecutar este programa, puedes divertirte animando a diferentes superhéroes con un número aleatorio de ¡vivas! en cada ocasión. Observa cómo no conocemos el número de repeticiones con anticipación en ninguno de los dos bucles. Aquí tienes un ejemplo de la salida de este programa:

[illegible]

```

Ingresa el nombre de un superhéroe (o 'salir' para terminar): Superman
¡Viva Superman!
¡Viva Superman!

Ingresa el nombre de un superhéroe (o 'salir' para terminar): salir

¡Adiós!

```

4. *Batalla Real*. Combinar estructuras de bucles y condicionales te permite programar lógica más compleja, como una batalla real de superhéroes. En el programa “Batalla Real”, el usuario ingresa dos héroes elegidos entre Iron-man, Captain America o Black Widow. El programa asigna niveles de poder aleatorios entre 1 y 100 a cada héroe y simula una batalla donde los héroes se turnan para golpearse, reduciendo el nivel de poder hasta 10 con cada golpe. La batalla continúa hasta que el nivel de poder de un héroe llega o cae por debajo de 0. El programa permite al usuario repetir la batalla si desea continuar:

```

1 import random
2
3 heroes = "iron-man/captain america/black widow"
4 while True:
5     hero1 = input("\nIngresa el nombre del primer héroe (o 'salir' para terminar): ")
6     if hero1 == "salir":
7         break
8
9     hero2 = input("Ingresa el nombre del segundo héroe (o 'salir' para terminar): ")
10    if hero2 == "salir":
11        break
12
13    if hero1.lower() not in heroes or hero2.lower() not in heroes:
14        print('Nombres de héroe no válidos. Solo se permiten "Iron-man", "Captain America" o "Black Widow". Por favor, intenta nuevamente.')
15        continue
16
17    poder1 = random.randint(1, 100)
18    poder2 = random.randint(1, 100)
19
20    print(f"\n¡Que comience la batalla entre {hero1} (nivel de poder: {poder1}) y {hero2} (nivel de poder: {poder2})!")
21

```

```

22 while poder1 > 0 and poder2 > 0:
23     golpe1 = random.randint(1, 10)
24     golpe2 = random.randint(1, 10)
25     poder1 -= golpe2
26     poder2 -= golpe1
27
28     print(f"\n{hero1} golpea a {hero2} con poder {golpe1} ¡Nivel de poder de {
29     hero2}: {poder2}!")
30     print(f"{hero2} contraataca y golpea a {hero1} con poder {golpe2} ¡Nivel de
31     poder de {hero1}: {poder1}!")
32
33 if poder1 <= 0 and poder2 <= 0:
34     print("\nEs un empate! Ambos héroes quedaron noqueados.")
35 elif poder1 <= 0:
36     print(f"\n{hero2} es el ganador!")
37 elif poder2 <= 0:
38     print(f"\n{hero1} es el ganador!")
39
40 print("\n¡Gracias por jugar a Batalla Real de Superhéroes! ¡Nos vemos la próxima
41 vez!")

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2p8z9yap>

Observa cómo en este código, se utilizan bucles anidados para controlar la progresión de la batalla y las repeticiones entre batallas. El bucle externo permite que el usuario continúe o salga del programa, mientras que el bucle interno simula la batalla en sí, con los héroes turnándose y reduciendo los niveles de poder del otro. Las condicionales anidadas se utilizan para validar la entrada del usuario, determinar el ganador/perdedor de la batalla y controlar la finalización del programa. Aseguran que el programa finalice cuando el usuario ingresa “salir” y definen la lógica para obtener el resultado de la batalla. Aquí tienes un ejemplo de la salida de este programa:

```

Ingresa el nombre del primer héroe (o 'salir' para terminar): Black Widow
Ingresa el nombre del segundo héroe (o 'salir' para terminar): Captain America

¡Que comience la batalla entre Black Widow (nivel de poder: 95) y Captain
America (nivel de poder: 72)!

Black Widow golpea a Captain America con poder 9 ¡Nivel de poder de Captain
America: 63!
Captain America contraataca y golpea a Black Widow con poder 8 ¡Nivel de poder
de Black Widow: 87!

```

Black Widow golpea a Captain America con poder 5 ¡Nivel de poder de Captain America: 58!

Captain America contraataca y golpea a Black Widow con poder 8 ¡Nivel de poder de Black Widow: 79!

Black Widow golpea a Captain America con poder 8 ¡Nivel de poder de Captain America: 50!

Captain America contraataca y golpea a Black Widow con poder 7 ¡Nivel de poder de Black Widow: 72!

Black Widow golpea a Captain America con poder 10 ¡Nivel de poder de Captain America: 40!

Captain America contraataca y golpea a Black Widow con poder 7 ¡Nivel de poder de Black Widow: 65!

Black Widow golpea a Captain America con poder 6 ¡Nivel de poder de Captain America: 34!

Captain America contraataca y golpea a Black Widow con poder 5 ¡Nivel de poder de Black Widow: 60!

Black Widow golpea a Captain America con poder 4 ¡Nivel de poder de Captain America: 30!

Captain America contraataca y golpea a Black Widow con poder 5 ¡Nivel de poder de Black Widow: 55!

Black Widow golpea a Captain America con poder 3 ¡Nivel de poder de Captain America: 27!

Captain America contraataca y golpea a Black Widow con poder 9 ¡Nivel de poder de Black Widow: 46!

Black Widow golpea a Captain America con poder 9 ¡Nivel de poder de Captain America: 18!

Captain America contraataca y golpea a Black Widow con poder 3 ¡Nivel de poder de Black Widow: 43!

Black Widow golpea a Captain America con poder 8 ¡Nivel de poder de Captain America: 10!

Captain America contraataca y golpea a Black Widow con poder 5 ¡Nivel de poder de Black Widow: 38!

Black Widow golpea a Captain America con poder 7 ¡Nivel de poder de Captain

```

America: 3!
Captain America contraataca y golpea a Black Widow con poder 10 ¡Nivel de
poder de Black Widow: 28!

Black Widow golpea a Captain America con poder 3 ¡Nivel de poder de Captain
America: 0!
Captain America contraataca y golpea a Black Widow con poder 10 ¡Nivel de
poder de Black Widow: 18!

¡Black Widow es la ganadora!

Ingresa el nombre del primer héroe (o 'salir' para terminar): salir
¡Gracias por jugar a Batalla Real de Superhéroes! ¡Nos vemos la próxima vez!
```

4.6 ¡Manos a la obra!

Sugiere algunos ejercicios de codificación divertidos y atractivos para practicar las sentencias if/else, if/elif/else y match/case, así como el operador ternario, incluyendo descripciones de problemas y ejemplos de salidas esperadas.⁷²

1. **Traductor de Emojis:** Escribe un programa que traduzca un emoticón de texto a su significado correspondiente. Usa una sentencia `match/case` para determinar los significados.

Ejemplo:

```

Entrada: ":)"
Salida: "Carita feliz"
```

2. **Calculadora de Calificaciones:** Crea un programa que calcule y muestre la calificación en letra basada en una calificación numérica dada. Usa declaraciones `if/elif/else` para asignar la calificación en letra apropiada.

Ejemplo:

```

Entrada: 85
Salida: B
```

3. **Pedido de Pizza:** Escribe un programa que tome la entrada del usuario para el tamaño y los ingredientes de una pizza y calcule el costo total. Usa declaraciones `if/elif/else` y el operador `in` para determinar el costo según el tamaño y los ingredientes.

Ejemplo:

Entrada: Tamaño - Mediano, Ingredientes - Pepperoni, Champiñones
Salida: Costo total: \$12.50

4. **Juego de Piedra, Papel o Tijera:** Implementa un sencillo juego de piedra, papel o tijera por consola. Usa declaraciones `match/case` para determinar el ganador según las elecciones de los jugadores.

Ejemplo:

Entrada: Jugador 1 - Piedra, Jugador 2 - Tijera
Salida: ¡Jugador 1 gana!

5. **Asesor de Ropa según el Clima:** Crea un programa que aconseje al usuario sobre qué ropa usar según las condiciones climáticas. Usa declaraciones `match/case` para proporcionar recomendaciones de ropa apropiadas.

Ejemplo:

Entrada: Clima - Soleado, Temperatura - 28°C
Salida: ¡Está soleado! Usa una camiseta y pantalones cortos.

6. **Recomendador de Música según Grupo de Edad:** Escribe un programa que clasifique a una persona en diferentes grupos de edad según su edad y recomiende bandas de música apropiadas. Usa declaraciones `if/elif/else` para categorizar el grupo de edad.

Ejemplo:

Entrada: Edad - 45
Salida: Escucha música de: Nirvana, R.E.M

7. **Comparaciones de Edades:** Crea un programa que compare tres edades para determinar quién es el más viejo y quién es el más joven. Usa declaraciones `if/elif/else` para realizar las comparaciones.

Ejemplo:

Entrada: Edades - 10, 5, 8
Salida: Más viejo: 10, Más joven: 5

Estos ejercicios proporcionan escenarios prácticos donde se pueden aplicar declaraciones `if/else`, `if/elif/else`, `match/case` y el operador ternario. Te ayudarán a reforzar la comprensión de las declaraciones condicionales y la toma de decisiones en la programación.

Sugiere algunos ejercicios de codificación divertidos y atractivos para practicar bucles for y while, incluyendo descripciones de problemas y ejemplos de salidas esperadas.⁷³

1. **Temporizador de Cuenta Regresiva:** Crea un programa que acepte un número de entrada y cuente hacia abajo desde ese número hasta uno, mostrando cada número en una nueva línea con una pausa de un segundo entre cada número.

Ejemplo:

```
Entrada: 5
Salida:
5 (espera un segundo...)
4 (espera un segundo...)
3 (espera un segundo...)
2 (espera un segundo...)
1 (espera un segundo...)
¡Listo!
```

2. **Tabla de Repetición:** Crea un programa que genere la tabla de repetición para un número dado. El programa debe mostrar la tabla del 1 al 10.

Ejemplo:

```
Entrada: 7
Salida:
7 x 1 = 7
7 x 2 = 77
7 x 3 = 777
7 x 4 = 7777
...
7 x 10 = 7777777777
```

3. **Calculadora de Factoriales:** Escribe un programa que calcule el factorial de un número dado utilizando un bucle **while**. El factorial de un número es el producto de todos los enteros positivos menores o iguales a ese número. El programa debe mostrar toda la secuencia de multiplicación.

Ejemplo:

```
Entrada: 5
Salida: 5! = 5 x 4 x 3 x 2 x 1 = 120
```

4. **Juego Adivina el Número:** Crea un programa que genere un número aleatorio entre 1 y 100. El usuario debe adivinar el número, y el programa debe proporcionar indicaciones (mayor o menor) hasta que se adivine el número correcto. Usa una sentencia `while/if` anidada y muestra un mensaje con el número de intentos utilizados.

Ejemplo:

```
Salida:
Adivina un número entre 1 y 100: 50
¡Demasiado alto!
Adivina un número entre 1 y 100: 25
¡Demasiado bajo!
Adivina un número entre 1 y 100: 40
¡Felicidades! Adivinaste el número en 3 intentos.
```

5. **Suma de Dígitos:** Escribe un programa que calcule la suma de los dígitos de un número dado utilizando un bucle `while`.

Ejemplo:

```
Entrada: 1234
Salida: 10 (ya que 1 + 2 + 3 + 4 = 10)
```

6. **Piedra, Papel o Tijera vs Computadora (2 de 3):** Implementa un juego de piedra, papel o tijera basado en texto en el que se jueguen 2 de 3 rondas (es decir, el primero en ganar dos rondas). Usa declaraciones `match/case` anidadas dentro de un bucle con una cláusula `break` para determinar el ganador según las elecciones de los jugadores en toda la partida.

Ejemplo:

```
Salida:
Elige tu opción: Piedra.
La computadora juega: Tijera
Resultado: ¡Jugador 1 gana!

Elige tu opción: Tijera.
La computadora juega: Tijera
Resultado: ¡Es un empate!

Elige tu opción: Papel.
La computadora juega: Piedra
Resultado: ¡Jugador 1 gana!
```

Ganador: ¡Jugador 1 es el ganador general!

7. **Adivinador de Edad del Usuario:** Crea un programa que adivine la edad del usuario haciendo una serie de preguntas tipo si/no. El programa debe ir reduciendo el rango de edad posible con cada pregunta hasta llegar a la edad correcta. El programa debe hacer suposiciones guiadas por las respuestas del usuario a las preguntas.

Ejemplo:

```
Salida:
¿Tienes más de 50 años? (sí/no): no
¿Tienes más de 25 años? (sí/no): sí
¿Tienes más de 37 años? (sí/no): no
¿Tienes más de 31 años? (sí/no): sí
¿Tienes más de 34 años? (sí/no): no
¿Tienes más de 32 años? (sí/no): sí
¿Tienes más de 33 años? (sí/no): sí

¡Ya lo sé! ¡Tienes 34 años!
```

Sugiere algunos ejercicios de programación divertidos e interesantes para practicar estructuras de bucles anidados, incluyendo descripciones de problemas y ejemplos de resultados esperados.⁷⁴

1. **Números Primos:** Escribe un programa que genere números primos hasta un límite dado utilizando un bucle `while/for` anidado. Un número primo es aquel que solo es divisible por 1 y por sí mismo.

Ejemplo:

```
Entrada: 20
Salida: 2, 3, 5, 7, 11, 13, 17, 19
```

2. **Impresión de Patrones:** Crea un programa que imprima varios patrones utilizando bucles anidados. Por ejemplo, puedes imprimir un triángulo, un cuadrado o incluso un patrón de diamante. Utiliza bucles anidados para controlar el número de filas y columnas.

Ejemplo:

```
Salida:
Patrón de Triángulo:
```

```
*
* *
* * *
* * * *
* * * * *
```

Patrón de Cuadrado:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Patrón de Diamante:

```
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * *
* * * * *
 * * * *
  * * *
   * *
    *
     *
    *
   * *
  * * *
 * * * *
* * * * *
 * * * *
  * * *
   * *
    *
     *
    *
   * *
  * * *
 * * * *
* * * * *
```

3. **Tabla de Multiplicar:** Escribe un programa que genere una tabla de multiplicar del 1 al 10. Utiliza bucles anidados para iterar sobre filas y columnas y calcular el producto.

Ejemplo:

```
Salida:
Tabla de Multiplicar:
1  2  3  4  5  6  7  8  9  10
2  4  6  8  10 12 14 16 18 20
3  6  9  12 15 18 21 24 27 30
...
10 20 30 40 50 60 70 80 90 100
```

4. **Calculadora de Notas:** Crea un programa que calcule el promedio de calificaciones para una clase. Solicita al usuario el número de estudiantes y el número de materias. Utiliza bucles anidados para ingresar y calcular las calificaciones, y luego muestra el promedio de notas.

Ejemplo:

```
Salida:
Ingresa el número de estudiantes: 3
Ingresa el número de materias: 4

Ingresa las calificaciones para el estudiante 1:
Materia 1: 85
Materia 2: 90
Materia 3: 80
Materia 4: 95

Ingresa las calificaciones para el estudiante 2:
Materia 1: 75
Materia 2: 70
Materia 3: 85
Materia 4: 80

Ingresa las calificaciones para el estudiante 3:
Materia 1: 95
Materia 2: 90
Materia 3: 92
Materia 4: 88

Promedio de Notas: 85.83
```

5. **Carrera de Letras:** Crea un programa que simule una carrera entre dos letras definidas por el usuario durante 5 movimientos consecutivos. Cada competidor puede moverse hacia la derecha hasta 5 pasos aleatorios en cada movimiento. El programa debe mostrar las posiciones de las letras después de cada movimiento y determinar al ganador al final del último movimiento utilizando el operador ternario. Puedes utilizar bucles anidados para controlar los movimientos de la carrera y los pasos de los competidores, respectivamente.

Ejemplo:

```
Entrada: A, Z
Salida:
[Movimiento 1]
..A
....Z
```

```

[Movimiento 2]
.....A
.....Z

[Movimiento 3]
.....A
.....Z

[Movimiento 4]
.....A
.....Z

[Movimiento 5]
.....A
.....Z

¡Es un empate!

```

6. **Juego FizzBuzz:** Escribe un programa que imprima números del 1 al 100. Para los múltiplos de 3, imprime "Fizz".^{en} lugar del número. Para los múltiplos de 5, imprime "Buzz". Para los números que son múltiplos de 3 y 5, imprime "FizzBuzz". Utiliza estructuras anidadas de `for/if/else` para verificar las condiciones.

Ejemplo:

```

Salida:
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz

```

Sugiere el problema de escribir un programa de calculadora basado en consola que tome dos números y una operación seleccionada de un menú (decorado con un marco), calcule y muestre el resultado, y pregunte al usuario si quiere continuar. Incluye un ejemplo de la salida esperada.⁷⁵

Calculadora Científica en Consola:

Escribe un programa de calculadora basado en consola que permita al usuario realizar operaciones aritméticas. La calculadora debe proporcionar un menú con una selección de las siguientes operaciones:

1. Suma (+): Sumar dos números.
2. Resta (-): Restar el segundo número del primer número.
3. Multiplicación (*): Multiplicar los dos números.
4. División (/): Dividir el primer número por el segundo número.
5. Exponente (^): Elevar el primer número a la potencia del segundo número.
6. Raíz Cuadrada (**sqrt**): Calcular la raíz cuadrada de un número.
7. Logaritmo (**log**): Calcular el logaritmo en base 10 de un número.

Ejemplo de salida:

```
*****
*           MENÚ DE CALCULADORA           *
*****
*  1. Suma                                *
*  2. Resta                                *
*  3. Multiplicación                       *
*  4. División                             *
*  5. Exponente                           *
*  6. Raíz Cuadrada                        *
*  7. Logaritmo                            *
*  8. Salir                               *
*****

Selecciona una operación (1-8): 3

Ingresa el primer número: 5
Ingresa el segundo número: 7

Resultado: 5 * 7 = 35
```



```

*****
*           MENÚ DE CALCULADORA           *
*****
*  1. Suma                                *
*  2. Resta                                *
*  3. Multiplicación                       *
*  4. División                             *
*  5. Exponente                           *
*  6. Raíz Cuadrada                        *
*  7. Logaritmo                            *
*  8. Salir                               *
*****
Selecciona una operación (1-8): 8

¡Adiós!

```

Implementar este problema te permitirá practicar el manejo de la entrada del usuario, el desarrollo de programas con menús de opciones, el uso de funciones matemáticas del módulo `math` y la implementación de estructuras de control anidadas. ¡Diviértete construyendo tu calculadora científica!

Zona de Contenedores

5

Explica qué es una estructura de contenedor, luego enumera todas las estructuras de contenedor disponibles en Python y proporciona una breve descripción para cada una, utilizando analogías de la vida real y un tono amigable.⁷⁶

Una estructura de contenedor en Python es una estructura de datos capaz de contener múltiples valores u objetos, organizados bajo un solo nombre. Permite organizar y almacenar datos de manera estructurada. Cada estructura de contenedor tiene sus propias características únicas y es adecuada para diferentes propósitos.

1. **Listas:** Piensa en una bolsa de compras donde puedes poner varios artículos. Es una colección ordenada de elementos que pueden ser de diferentes tipos.
2. **Tuplas:** Imagina un sobre sellado que contiene múltiples piezas de información. Una tupla es similar a una lista, pero es inmutable, lo que significa que no puede modificarse una vez creada.
3. **Conjuntos:** Visualiza un cajón lleno de elementos únicos. Un conjunto es una colección desordenada de elementos únicos.
4. **Cadenas de caracteres:** Considera una cadena como una oración o una secuencia de caracteres. Es un contenedor inmutable que contiene una colección de símbolos.
5. **Diccionarios:** Imagina un diccionario en la vida real donde se almacenan palabras y sus definiciones. Un diccionario es una estructura de datos de pares clave-valor.

Estas estructuras de contenedores brindan flexibilidad para organizar y manipular datos según tus necesidades específicas. Al elegir la estructura de contenedor adecuada, puedes almacenar y procesar datos de manera eficiente en tus programas de Python.

De manera que, demos un vistazo más de cerca a cada una de ellas.

5.1 Listas

Explica todo lo que necesito saber sobre listas en Python de una manera divertida y creativa, usando ejemplos lúdicos.⁷⁷

¡Claro! ¡Vamos a embarcarnos en una aventura divertida en el mundo de las listas de Python! Imagina que eres un mago con una mochila mágica que puede contener todo tipo de objetos fantásticos. En Python, las listas son como tu mochila mágica, donde puedes guardar, agregar, eliminar y recuperar elementos a tu conveniencia.

1. **Creando Listas:** Para crear una lista, imagínala como tu mochila mágica. Puedes poner cualquier cosa que desees en ella, como brillantes gemas, poderosas pociones o pergaminos encantados. Solo usa corchetes `[]` y separa los elementos con comas.
2. **Indexación:** Cada elemento en tu mochila mágica tiene un número especial llamado índice. Es como una etiqueta que te ayuda a encontrar cosas rápidamente. ¡Recuerda, los magos comienzan a contar desde 0! Así que el primer elemento está en el índice 0, el segundo en el índice 1, y así sucesivamente. Puedes acceder a los elementos de tu lista utilizando estos índices.
3. **Actualización y Adición de Elementos:** ¡Tu mochila mágica es versátil! Puedes agregar nuevos elementos o cambiar los existentes en cualquier momento. Imagina encontrar un objeto raro durante tus aventuras y reemplazarlo con uno aún más poderoso. Puedes usar el índice para actualizar los elementos en tu lista.
4. **Seccionamiento de Listas:** A veces quieres presumir tu colección de objetos mágicos a otros magos. El corte de listas te permite tomar un subconjunto de tu lista y mostrarlo con orgullo. Es como mostrarles un vistazo de tu extraordinario inventario.
5. **Longitud y Pertenencia:** ¡Tu mochila mágica puede hacer más! Puede decirte cuántos elementos tienes usando la función `len()`. También puedes preguntar si un elemento en particular está en tu lista usando la palabra clave `in`. Es como verificar si un artefacto mágico específico está en tu posesión.
6. **Operaciones con Listas:** ¡Puedes realizar operaciones mágicas en tu lista! ¿Quieres combinar dos listas? Solo usa el operador `+` para fusionarlas en una súper lista poderosa. Incluso puedes multiplicar tu lista para crear múltiples copias de sus contenidos. Es como multiplicar tus objetos mágicos para reproducirlos.
7. **Métodos de Listas:** Tu mochila mágica viene con hechizos prácticos conocidos como métodos. Estos métodos te ayudan a ordenar tus elementos, agregar nuevos, eliminar los no deseados o incluso invertir el orden de tu lista. Son como poderes especiales que facilitan la gestión de tu lista.

Así que ¡agarra tu mochila mágica (es decir, tu lista de Python) y comienza a llenarla con todo tipo de maravillas! Recuerda, ¡cuanto más creativo seas, más divertido será!

De acuerdo. Muéstrame un ejemplo de código que ilustre cada uno de los elementos mencionados anteriormente. Incluye comentarios en cada bloque de código para explicar su propósito. Utiliza si quieres el tema de superhéroes contra villanos.⁷⁸

¡Por supuesto! Aquí tienes ejemplos de código que ilustran cada uno de los elementos mencionados anteriormente, utilizando el tema de superhéroes contra villanos. Cada bloque de código está acompañado de comentarios que explican el ítem correspondiente, junto con la salida en la consola.

```

1 # Creando Listas
2 superheroes = [] # Comienza con una lista vacía
3 print("¡Tu mochila mágica de superhéroes está lista!")
4 print(f"Lista de superhéroes: {superheroes}")

```

```

¡Tu mochila mágica de superhéroes está lista!
Lista de superhéroes: []

```

```

1 # Poblar la Lista
2 superheroes = ["Spider-Man", "Iron Man", "Capitán América"]
3 print("\n¡Ahora tu mochila mágica de superhéroes está llena!")
4 print(f"Lista de superhéroes: {superheroes}")

```

```

¡Ahora tu mochila mágica de superhéroes está llena!
Lista de superhéroes: ['Spider-Man', 'Iron Man', 'Capitán América']

```

```

1 # Indexación
2 print("\nEjemplo de Indexación:")
3 print(f"El primer superhéroe es: {superheroes[0]}")
4 print(f"El segundo superhéroe es: {superheroes[1]}")
5 print(f"El último superhéroe es: {superheroes[-1]}")

```

```

Ejemplo de Indexación:
El primer superhéroe es: Spider-Man
El segundo superhéroe es: Iron Man
El último superhéroe es: Capitán América

```

```

1 # Actualización y Adición de Elementos
2 print("\nEjemplo de Actualización y Agregado de Elementos:")
3 superheroes[1] = "Viuda Negra" # Actualizar un elemento en el índice 1
4 print("Lista actualizada de superhéroes:", superheroes)
5
6 superheroes.append("Hulk") # Agregar un nuevo elemento al final de la lista
7 print("Lista extendida de superhéroes:", superheroes)

```

```

Ejemplo de Actualización y Adición de Elementos:
Lista actualizada de superhéroes: ['Spider-Man', 'Viuda Negra', 'Capitán América']
Lista extendida de superhéroes: ['Spider-Man', 'Viuda Negra', 'Capitán América',
'Hulk']

```

```

1 # Seccionamiento de Listas
2 print("\nEjemplo de Seccionamiento de Listas:")
3 villains = ["Duende Verde", "Loki", "Calavera Roja", "Thanos", "Veneno"]
4 print("Lista de villanos: ", villains)
5 print("Los primeros tres villanos son:", villains[:3])
6 print("Los últimos dos villanos son:", villains[-2:])

```

Ejemplo de Seccionamiento de Listas:

Lista de villanos: ['Duende Verde', 'Loki', 'Calavera Roja', 'Thanos', 'Veneno']

Los primeros tres villanos son: ['Duende Verde', 'Loki', 'Calavera Roja']

Los últimos dos villanos son: ['Thanos', 'Veneno']

```

1 # Longitud y Pertenencia
2 print("\nEjemplo de Longitud y Pertenencia:")
3 print("Número de superhéroes:", len(superheroes))
4 print("Número de villanos:", len(villains))
5
6 hero = input("Ingresa el nombre de un superhéroe para verificar si está en la lista: ")
7
8 if hero in superheroes:
9     print(f"Sí, i{hero} está en tu mochila mágica!")
10 else:
11     print(f"Lo siento, {hero} no está en tu mochila mágica.")

```

Ejemplo de Longitud y Pertenencia:

Número de superhéroes: 4

Número de villanos: 5

Ingresa el nombre de un superhéroe para verificar si está en la lista: Superman

Lo siento, Superman no está en tu mochila mágica.

```

1 # Operaciones con Listas
2 print("\nEjemplo de Operaciones con Listas:")
3 all_characters = superheroes + villains # Combinar dos listas
4 print("Todos los personajes en el universo de superhéroes:", all_characters)
5
6 duplicate_villains = villains * 2 # Crear múltiples copias de la lista
7 print("Villanos duplicados:", duplicate_villains)

```

Ejemplo de Operaciones con Listas:

Todos los personajes en el universo de superhéroes: ['Spider-Man', 'Viuda Negra', 'Capitán América', 'Hulk', 'Duende Verde', 'Loki', 'Calavera Roja', 'Thanos', 'Veneno']

Villanos duplicados: ['Duende Verde', 'Loki', 'Calavera Roja', 'Thanos', 'Veneno', 'Duende Verde', 'Loki', 'Calavera Roja', 'Thanos', 'Veneno']

```

1 # Métodos de Listas
2 print("\nEjemplo de Métodos de Listas:")
3 print("Lista original de superhéroes:", superheroes)
4 superheroes.sort() # Ordenar la lista en orden alfabético
5 print("Lista ordenada de superhéroes:", superheroes)
6
7 superheroes.remove("Capitán América") # Eliminar un elemento específico de la lista
8 print("Lista actualizada de superhéroes:", superheroes)
9 superheroes.reverse() # Invertir el orden de la lista
10 print("Lista invertida de superhéroes:", superheroes)

```

Ejemplo de Métodos de Listas:

Lista original de superhéroes: ['Spider-Man', 'Viuda Negra', 'Capitán América', 'Hulk']

Lista ordenada de superhéroes: ['Viuda Negra', 'Capitán América', 'Hulk', 'Spider-Man']

Lista actualizada de superhéroes: ['Viuda Negra', 'Hulk', 'Spider-Man']

Lista invertida de superhéroes: ['Spider-Man', 'Hulk', 'Viuda Negra']

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2s4bz4sy>

*Interesante. Ahora demuéstrame cómo puedo usar bucles y condicionales con listas. Esta vez, utiliza ejemplos creativos relacionados con comida, aprovechando las características de aleatoriedad de Python.*⁷⁹

¡Por supuesto! Aquí tienes un ejemplo de cómo puedes usar bucles y condicionales con listas, utilizando un tema creativo relacionado con la comida que aprovecha las características de aleatoriedad de Python. El código demuestra diferentes formas de iterar sobre una lista de alimentos y aplicar sentencias condicionales para diversos escenarios relacionados con la comida. ¡Disfruta de los deliciosos ejemplos!

```

1 import random
2 foods = ["pizza", "hamburguesa", "sushi", "taco", "pasta"]
3

```

```

4 # Iterando sobre una lista
5 print("=== Menú ===")
6 for food in foods:
7     print(food)

```

```

=== Menú ===
pizza
hamburguesa
sushi
taco
pasta

```

```

1 # Bucle con condicionales
2 print("\n=== Opciones Vegetarianas ===")
3 for food in foods:
4     if food != "hamburguesa":
5         print(food)

```

```

=== Opciones Vegetarianas ===
pizza
sushi
taco
pasta

```

```

1 # Selección aleatoria de un alimento
2 print("\n=== Comida Aleatoria ===")
3 food_aleatorio = random.choice(foods)
4 print("¡Disfruta tu {food_aleatorio}!")

```

```

=== Comida Aleatoria ===
¡Disfruta tu pizza!

```

```

1 # Bucle con un toque aleatorio
2 print("\n=== Ruleta de Comida ===")
3 for i in range(3):
4     food_aleatorio = random.choice(foods)
5     if food_aleatorio == "sushi":
6         print(f"Ronda {i+1}: ¡Es tu día de suerte! ¡Obtuviste sushi!")
7     else:
8         print(f"Ronda {i+1}: Mejor suerte la próxima vez. Obtuviste {food_aleatorio}")

```



```

=== Ruleta de Comida ===
Ronda 1: Mejor suerte la próxima vez. Obtuviste taco
Ronda 2: ¡Es tu día de suerte! ¡Obtuviste sushi!
Ronda 3: Mejor suerte la próxima vez. Obtuviste pasta

```

```

1 # Precios aleatorios con una lista creada por comprensión (list comprehensions)
2 prices = [random.randint(2, 20) for _ in range(len(foods))]
3 print("\n=== Lista de Precios ===")
4 for food, price in zip(foods, prices):
5     print(f"{food}\t${price}")

```

```

=== Lista de Precios ===
pizza $7
hamburguesa $15
sushi $5
taco $10
pasta $14

```

```

1 # Listas Anidadas y Bucles Anidados
2 combos = [
3     ["sandwich", "papas fritas", "jugo"],
4     ["hamburguesa", "papas fritas", "gaseosa"],
5     ["taco", "fruta", "cerveza"]
6 ]
7
8 print("\n=== Combos de Comida ===")
9 for combo in combos:
10     print(f"\nCombinación del combo {combo[0].upper()}:")
11     for i, item in enumerate(combo):
12         print(f"Plato {i+1}: {item}")

```

```

=== Combos de Comida ===

Combinación del combo SANDWICH:
Plato 1: sandwich
Plato 2: papas fritas
Plato 3: jugo

Combinación del combo HAMBURGUESA:
Plato 1: hamburguesa
Plato 2: papas fritas
Plato 3: gaseosa

```

Combinación del combo TACO:

Plato 1: taco

Plato 2: fruta

Plato 3: cerveza

```

1 # Métodos y Operaciones con Listas
2 drinks = ["gaseosa", "jugo", "cerveza"]
3
4 print("\n=== Bebidas ===")
5 print("Lista original de bebidas:", drinks)
6 drinks.insert(1, "agua")
7 print("Lista actualizada de bebidas:", drinks)
8
9 todo_junto = foods + drinks
10 print("\n=== Todo Junto ===")
11 print("Lista combinada de alimentos y bebidas:", todo_junto)

```

=== Bebidas ===

Lista original de bebidas: ['gaseosa', 'jugo', 'cerveza']

Lista actualizada de bebidas: ['gaseosa', 'agua', 'jugo', 'cerveza']

=== Todo Junto ===

Lista combinada de alimentos y bebidas: ['pizza', 'hamburguesa', 'sushi', 'taco',
'pasta', 'gaseosa', 'agua', 'jugo', 'cerveza']

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/mumh24r8>

*¡Espera un momento! En los ejemplos anteriores, usas algunos trucos interesantes de los que no estoy al tanto, como `random.choice()`, `zip()`, `enumerate()` e `insert()`. ¿Podrías explicármelos, por favor?*⁸⁰

¡Por supuesto! Vamos a repasar cada uno de esos trucos interesantes que se utilizaron en los ejemplos.

► **random.choice():**

- Esta es una función del módulo `random` en la biblioteca estándar de Python.
- Te permite seleccionar aleatoriamente un elemento de una lista dada.
- En el ejemplo, `random.choice(foods)` selecciona un elemento aleatorio de la lista `foods`.

► **zip():**

- `zip()` es otra función incorporada de Python que combina múltiples iterables (listas, tuplas, etc.) en un solo objeto iterable.

- Devuelve pares de elementos correspondientes de los iterables de entrada.
- En el ejemplo, `for food, price in zip(foods, prices):`, la función `zip()` se utiliza para iterar simultáneamente sobre dos listas, `foods` y `prices`, y las variables `food` y `price` contienen los elementos individuales de cada lista, para cada iteración.

► **enumerate():**

- `enumerate()` es una función incorporada de Python que te permite iterar sobre una lista (o cualquier iterable) y, al mismo tiempo, obtener tanto los elementos como sus índices.
- Devuelve tuplas que contienen el índice y el elemento actual del iterable.
- En el ejemplo, `for i, item in enumerate(combo):`, se utiliza para iterar sobre la lista `combo` y obtener tanto el índice `i` como el elemento `item` en cada iteración.

► **insert():**

- `insert()` es un método de lista que te permite insertar un elemento en una posición específica de la lista.
- Toma dos argumentos: el índice donde se insertará el elemento y el propio elemento que se va a insertar.
- En el ejemplo, `drinks.insert(1, "agua")`, se utiliza para insertar la cadena “agua” en la posición 1 de la lista `drinks`, desplazando los elementos existentes hacia la derecha.

Estos trucos son potentes y comúnmente utilizados en Python para mejorar la funcionalidad y hacer que el código sea más conciso y legible. Pueden ser muy útiles en varios escenarios, como generar selecciones aleatorias, iterar sobre listas con índices, o iterar varias listas juntas. ¡Ahora tienes algunos trucos de reserva en tu mochila mágica!

5.2 Tuplas

Explícame todo lo que necesito saber sobre las tuplas en Python de una manera divertida y creativa, usando ejemplos lúdicos.⁸¹

¡Por supuesto! Sumerjámonos en el fascinante mundo de las tuplas en Python. Imagina que estás organizando una gran cena para tus amigos y quieres llevar un registro de los asientos. ¡Las tuplas te pueden ayudar con eso! Piensa en una tupla como una configuración especial de mesa donde cada invitado tiene un asiento designado. Esto es lo que necesitas saber sobre las tuplas:

1. **Alineación Inmutable:** Una vez que estableces los asientos utilizando una tupla, quedan fijos. No puedes cambiar ni intercambiar los asientos sin crear una nueva tupla. Así como en una elegante cena, una vez que los invitados están sentados, no puedes moverlos.

2. **Colocación Ordenada:** Las tuplas preservan el orden de los elementos. Así como planeas meticulosamente la ubicación de tus invitados, las tuplas aseguran que el orden de los elementos se mantenga intacto. El primer elemento siempre estará en el primer asiento, el segundo elemento en el segundo asiento, y así sucesivamente.
3. **Combinación Variada:** Las tuplas te permiten tener una disposición de asientos diversa. Cada asiento puede contener un tipo de dato diferente, como cadenas, números o incluso otras tuplas. Puedes combinar y mezclar para crear estructuras de asientos complejas. Es como organizar una fiesta donde los asientos son sillones, butacas, sofás, etc.
4. **Eficiente y Liviana:** Las tuplas son livianas y eficientes. Ocupan menos memoria en comparación con otras estructuras de datos. Es como tener sillas compactas pero elegantes, que no ocupan mucho espacio en la mesa.
5. **Truco de Desempaquetado:** Un aspecto útil de las tuplas es el desempaquetado de tuplas. Puedes asignar fácilmente los elementos de una tupla a variables individuales en una sola línea de código. Es como revelar la disposición de los asientos entregando tarjetas de nombres a cada invitado.

Recuerda que las tuplas son más adecuadas para situaciones en las que necesitas una colección ordenada de elementos que no debe cambiar. Ya sea una disposición de asientos, coordenadas en un mapa o cualquier otro escenario donde la inmutabilidad y el orden sean importantes, ¡las tuplas son ideales para ello!

*Ok. Ahora veamos algunos ejemplos de código para ver las tuplas en acción. ¿Por qué no usamos bandas de rock esta vez para ilustrar los elementos que mencionaste anteriormente?*⁸²

¡Claro! Veamos algunos ejemplos de código usando bandas de rock para ilustrar los diferentes aspectos de las tuplas. Exploraremos cada elemento mencionado anteriormente de una manera divertida y creativa.

1. **Alineación Inmutable:** Supongamos que estás organizando un concierto de rock, puedes usar una tupla para definir la alineación de las bandas. Una vez que la alineación esté establecida, permanecerá fija durante todo el concierto. Por ejemplo:

```
1 alineacion = ("The Beatles", "Led Zeppelin", "AC/DC", "Guns N' Roses", "Queen")
2 print(alineacion)
3
4 # Descomentar la línea de abajo generará un error
5 # alineacion[0] = "Coldplay"
```

Una vez que se establece la alineación utilizando una tupla, no se puede cambiar. Intentar modificar un elemento de la tupla generará un error, ya que las tuplas son inmutables, al igual que intentar reemplazar una banda después de que ha comenzado el concierto.

```
=== Alineación Inmutable ===
('The Beatles', 'Led Zeppelin', 'AC/DC', 'Guns N' Roses', 'Queen')
```

2. **Colocación Ordenada:** Al planificar un concierto, quieres asegurarte de que las bandas actúen en un orden específico. Las tuplas preservan el orden de los elementos. Veamos:

```
1 alineacion = ("The Beatles", "Led Zeppelin", "AC/DC", "Guns N' Roses", "Queen")
2
3 for banda in alineacion:
4     print("Siguiente banda:", banda)
```

En este ejemplo, las bandas actúan en el orden definido por la tupla. El bucle itera sobre la alineación y se imprime cada banda en el orden correcto.

```
=== Colocación Ordenada ===
Siguiente banda: The Beatles
Siguiente banda: Led Zeppelin
Siguiente banda: AC/DC
Siguiente banda: Guns N' Roses
Siguiente banda: Queen
```

3. **Combinación Variada:** Las bandas de rock pueden tener alineaciones diversas con miembros que desempeñan diferentes roles. Las tuplas te permiten crear estructuras complejas combinando diferentes tipos de datos. Ilustremos este aspecto con los Beatles:

```
1 john = ("John Lennon", 1940, False)
2 paul = ("Paul McCartney", 1942, True)
3 george = ("George Harrison", 1943, False)
4 ringo = ("Ringo Starr", 1940, True)
5
6 beatles = (john, paul, george, ringo)
7
8 for integrante in beatles:
9     print("\nIntegrante de los Beatles:", integrante[0])
10    print("Año de nacimiento:", integrante[1])
11    print("¿Está vivo?", integrante[2])
```

En este ejemplo, cada integrante de los Beatles está representado por una tupla que contiene el nombre, el año de nacimiento y un valor booleano que indica si están vivos o no. La tupla `beatles` combina las tuplas individuales de los integrantes para formar la banda completa.

```

=== Combinación Variada ===

Integrante de los Beatles: John Lennon
Año de nacimiento: 1940
¿Está vivo? False

Integrante de los Beatles: Paul McCartney
Año de nacimiento: 1942
¿Está vivo? True

Integrante de los Beatles: George Harrison
Año de nacimiento: 1943
¿Está vivo? False

Integrante de los Beatles: Ringo Starr
Año de nacimiento: 1940
¿Está vivo? True

```

4. **Efficiente y Ligera:** Las tuplas son livianas y ocupan menos memoria en comparación con otras estructuras de datos, como listas. Veamos un ejemplo:

```

1 beatles_lista = ["John Lennon", "Paul McCartney", "George Harrison", "Ringo Starr"]
2 beatles_tupla = ("John Lennon", "Paul McCartney", "George Harrison", "Ringo Starr")
3
4 print("Tamaño de la lista:", beatles_lista.__sizeof__())
5 print("Tamaño de la tupla:", beatles_tupla.__sizeof__())

```

En este ejemplo, usamos la función `__sizeof__()` para determinar el uso de memoria de `beatles_lista` y `beatles_tupla`. Notarás que el tamaño de la tupla es menor en comparación con la lista.

```

=== Eficiente y Ligera ===

Tamaño de la lista: 72
Tamaño de la tupla: 56

```

5. **Truco de Desempaquetado:** El desempaquetado de tuplas te permite asignar los elementos de una tupla a variables individuales. Continuemos ilustrando con la banda de los Beatles:

```

1 banda = ("The Beatles", "John Lennon", "Paul McCartney", "George Harrison", "Ringo Starr")

```

```

2
3 nombre_banda, *integrantes = banda
4
5 print("Nombre de la banda:", nombre_banda)
6 print("Integrantes de la banda:", integrantes)

```

En este ejemplo, desempaquetamos la tupla `banda` en variables individuales. La primera variable `nombre_banda` contiene el nombre de la banda, y a la variable `integrantes` se le asigna una lista que contiene los integrantes individuales de la banda.

```

=== Truco de Desempaquetado ===
Nombre de la banda: The Beatles
Integrantes de la banda: ['John Lennon', 'Paul McCartney', 'George Harrison',
                          'Ringo Starr']

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2mprsujd>

Estos ejemplos de código demuestran los diversos aspectos de las tuplas utilizando analogías divertidas de bandas de rock. Las tuplas son versátiles y se pueden usar en diversos escenarios más allá del rock'n'roll.

¡Espera! Hay un asterisco () extraño antes de la variable “integrantes” en el ejemplo del truco de desempaquetado. ¿Qué significa eso exactamente?*⁸³

El asterisco (*) antes de una variable en el desempaquetado de tuplas se utiliza para recolectar cualquier elemento restante de la tupla, en una lista. Esto se conoce como sintaxis de *desempaquetado extendido iterable* en Python. En el ejemplo de los Beatles, el asterisco se utiliza para asignar los miembros restantes de la tupla de la banda a la variable `integrantes`. Esto permite flexibilidad en caso de que la cantidad de integrantes en la banda no se conozca de antemano.

Sin embargo, si conoces la cantidad de elementos restantes en la tupla, puedes asignarlos a variables individuales, como se muestra en la siguiente variación del ejemplo:

```

1 banda = ("The Beatles", "John Lennon", "Paul McCartney", "George Harrison", "Ringo Starr"
2         )
3 nombre_banda, vocalista, guitarrista, bajista, baterista = banda
4
5 print("Nombre de la banda:", nombre_banda)
6 print("Vocalista:", vocalista)
7 print("Guitarrista:", guitarrista)
8 print("Bajista:", bajista)
9 print("Baterista:", baterista)

```

```
Nombre de la banda:: The Beatles
Vocalista: John Lennon
Guitarrista: Paul McCartney
Bajista: George Harrison
Baterista: Ringo Starr
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2c5durbt>

5.3 Conjuntos

Explicame todo lo que necesito saber sobre los conjuntos en Python de una manera divertida y creativa, usando ejemplos lúdicos.⁸⁴

¡Por supuesto! Adentrémonos en el mágico mundo de los conjuntos en Python. Imagina que tienes una colección de criaturas mágicas únicas. Cada criatura posee habilidades y características distintas. ¡Eso es exactamente lo que representan los conjuntos!

1. **Colección Única:** Un conjunto es como un cofre del tesoro especial donde puedes almacenar una colección de elementos únicos. Es como tener un zoológico mágico donde ninguna criatura puede ser igual a otra. Cada criatura es única y diferente de las demás.
2. **Sin Duplicados:** Los conjuntos aseguran que no se permitan duplicados de criaturas. Si intentas agregar la misma criatura dos veces, el conjunto mágicamente mantendrá solo una instancia de ella. Es como si el conjunto tuviera un poder mágico para mantener su singularidad.
3. **Pruebas de Membresía Rápidas:** Los conjuntos son excelentes para responder preguntas como “¿Esta criatura ya está en la colección?”. Pueden realizar pruebas de membresía de forma rápida como un rayo. Es como tener un escáner mágico que detecta rápidamente si una criatura forma parte del conjunto o no.
4. **Operaciones Mágicas:** Los conjuntos también ofrecen poderosas operaciones mágicas que puedes realizar sobre ellos. Puedes combinar conjuntos para crear nuevos conjuntos, encontrar criaturas comunes entre conjuntos o incluso encontrar criaturas únicas que existen solo en un conjunto y no en otro.
5. **Mutable y Sin Orden:** Los conjuntos son mutables, lo que significa que puedes agregar o eliminar criaturas de ellos. Sin embargo, no tienen un orden particular. Es como tener una reunión mágica donde las criaturas pueden moverse libremente sin ningún arreglo fijo.
6. **Eliminación Mágica:** Puedes eliminar mágicamente criaturas de un conjunto según sus propiedades únicas. Solo tienes que decir el nombre de la criatura que deseas eliminar, y el conjunto hará su magia y eliminará esa criatura.

7. Uniendo Fuerzas: Los conjuntos te permiten unir fuerzas realizando operaciones de unión. Puedes combinar múltiples conjuntos y crear una gran colección de todas las criaturas únicas de todos ellos. Es como fusionar tribus mágicas en una gran reunión.

*Muy bien. Ahora veamos algunos ejemplos de código para ver los conjuntos en acción. Esta vez, usemos datos divertidos sobre los planetas para ilustrar los elementos que mencionaste anteriormente.*⁸⁵

¡Claro! Aquí tienes algunos ejemplos de código que demuestran los conjuntos utilizando datos divertidos sobre los planetas:

```

1 # Colección Única
2 print("\n=== Colecciones Únicas ===")
3 planetas = set(["Mercurio", "Venus", "Tierra", "Marte", "Júpiter", "Saturno", "Urano", "
    Neptuno"])
4 lunas = set([0, 0, 1, 2, 79, 82, 27, 14])
5 ubicaciones = set(["interna", "interna", "interna", "interna", "externa", "externa", "
    externa", "externa"])
6 galaxias = set(["Vía Láctea", "Vía Láctea", "Vía Láctea", "Vía Láctea", "Vía Láctea", "Vía
    Láctea", "Vía Láctea", "Vía Láctea"])
7
8 print(f"Planetas (valores únicos: {len(planetas)}):", planetas)
9 print(f"Lunas (valores únicos: {len(lunas)}):", lunas)
10 print(f"Ubicaciones (valores únicos: {len(ubicaciones)}):", ubicaciones)
11 print(f"Galaxias (valores únicos: {len(galaxias)}):", galaxias)

```

En este ejemplo, estamos creando conjuntos a partir de la lista de planetas y listas que contienen el número correspondiente de lunas, ubicaciones dentro del sistema solar y galaxias a las que pertenecen.

Para crear un conjunto en Python, puedes usar llaves ({}) o la función `set()`. Simplemente encierra los elementos que deseas incluir en el conjunto entre llaves o pásalos como argumentos a la función `set()`. Los conjuntos son colecciones no ordenadas de elementos únicos, lo que significa que los valores duplicados se eliminan automáticamente. Son útiles cuando deseas almacenar una colección de elementos sin un orden específico y asegurar la singularidad entre los elementos.

```

=== Colecciones Únicas ===
Planetas (valores únicos: 8): {'Urano', 'Tierra', 'Marte', 'Venus', 'Neptuno',
    'Júpiter', 'Saturno', 'Mercurio'}
Lunas (valores únicos: 7): {0, 1, 2, 14, 79, 82, 27}
Ubicaciones (valores únicos: 2): {'interna', 'externa'}
Galaxias (valores únicos: 1): {'Vía Láctea'}

```

```

1 # Sin Duplicados
2 print("\n=== Sin Duplicados ===")
3 planetas = {"Mercurio", "Venus", "Tierra", "Marte", "Júpiter", "Saturno", "Urano", "
    Neptuno"}
4 print('Antes de agregar "Tierra":', planetas)
5 planetas.add("Tierra") # Agregar un planeta duplicado no tiene efecto
6 print('Después de agregar "Tierra":', planetas)

```

```

=== Sin Duplicados ===
Antes de agregar "Tierra": {'Urano', 'Tierra', 'Marte', 'Venus', 'Neptuno',
    'Júpiter', 'Saturno', 'Mercurio'}
Después de agregar "Tierra": {'Urano', 'Tierra', 'Marte', 'Venus', 'Neptuno',
    'Júpiter', 'Saturno', 'Mercurio'}

```

```

1 # Pruebas de Membresía Rápidas
2 print("\n=== Pruebas de Membresía Rápidas ===")
3 print("¿Está la Tierra en los planetas? ", "Tierra" in planetas)
4 print("¿Está Plutón en los planetas? ", "Plutón" in planetas)

```

```

=== Pruebas de Membresía Rápidas ===
¿Está la Tierra en los planetas? True
¿Está Plutón en los planetas? False

```

```

1 # Operaciones Mágicas
2 print("\n=== Operaciones Mágicas ===")
3 gigantes_gaseosos = {"Júpiter", "Saturno"}
4 planetas_rocosos = {"Mercurio", "Venus", "Tierra", "Marte"}
5 sistemas_de_anillos = {"Saturno", "Urano", "Neptuno"}
6
7 comunes_gaseosos_rocosos = gigantes_gaseosos.intersection(planetas_rocosos) # Da como
    resultado un conjunto vacío
8
9 comunes_anillos_gaseosos = sistemas_de_anillos.intersection(gigantes_gaseosos)
10
11 print("Planetas que son gigantes gaseosos: ", gigantes_gaseosos)
12 print("Planetas con superficies rocosas: ", planetas_rocosos)
13 print("Planetas con sistemas de anillos: ", sistemas_de_anillos)
14 print("Planetas con gas y roca en común: ", comunes_gaseosos_rocosos)
15 print("Planetas con anillos y gas en común: ", comunes_anillos_gaseosos)

```

=== Operaciones Mágicas ===

Planetas que son gigantes gaseosos: {'Júpiter', 'Saturno'}

Planetas con superficies rocosas: {'Tierra', 'Marte', 'Mercurio', 'Venus'}

```
Planetas con sistemas de anillos: {'Neptuno', 'Saturno', 'Urano'}
```

Planetas con gas y roca en común: `set()`

Planetas con anillos y gas en común: {'Saturno'}

```
1 # Mutable y Sin Orden
2 print("\n=== Mutable y Sin Orden ===")
3 print("Antes de eliminar 'Urano':", planetas)
4 planetas.remove("Urano")
5 print("Después de eliminar 'Urano':", planetas)
6
7 # Eliminación Mágica
8 print("\n=== Eliminación Mágica ===")
9 print("Antes de descartar 'Plutón':", planetas)
10 planetas.discard("Plutón") # Eliminar un planeta que no existe no tiene efecto
11 print("Después de descartar 'Plutón':", planetas)
```

```

1 # Uniendo Fuerzas
2 print("\n=== Uniendo Fuerzas ===")
3 planetas_enanos = {"Plutón", "Eris", "Haumea"}
4 todos_los_planetas = planetas.union(planetas_enanos)
5 print("Planetas originales: ", planetas)
6 print("Planetas enanos: ", planetas_enanos)
7 print("Todos los planetas: ", todos_los_planetas)

```

```

=== Uniendo Fuerzas ===
Planetas originales: {'Tierra', 'Marte', 'Mercurio', 'Venus', 'Neptuno', 'Júpiter',
                      'Saturno'}
Planetas enanos: {'Haumea', 'Plutón', 'Eris'}
Todos los planetas: {'Urano', 'Tierra', 'Marte', 'Venus', 'Plutón', 'Neptuno',
                      'Eris', 'Júpiter', 'Saturno', 'Mercurio', 'Haumea'}

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/sfhaje>

Como puedes ver, con los conjuntos puedes crear una colección única de planetas, realizar operaciones como comprobar la membresía, encontrar planetas comunes y agregar o eliminar planetas del conjunto. Los conjuntos ofrecen una forma conveniente y eficiente de trabajar con colecciones de elementos únicos, como explorar el fascinante mundo de los planetas con sus características distintas.

5.4 Cadenas de caracteres

*Explícame todo lo que necesito saber sobre las cadenas de caracteres en Python de una manera divertida y creativa, usando ejemplos lúdicos.*⁸⁶

¡Claro! ¡Vamos a embarcarnos en un divertido viaje a través del mundo de las cadenas de texto en Python!

Imagina las cadenas de caracteres como una colección mágica de letras, símbolos y palabras que pueden ser manipuladas y transformadas para crear todo tipo de cosas emocionantes. Son como los coloridos bloques de construcción de texto en Python. También se conocen como *strings*.

1. **Mezcla y combina:** ¡Las cadenas de texto tienen súper poderes para mezclarse! Pueden combinarse usando el operador +, ¡como mezclar pociones para crear nuevos hechizos! Por ejemplo, imagina combinar las cadenas “Hola” y “Mundo” para crear el poderoso hechizo “HolaMundo” o “MundoHola” u “HolaHolaMundo”. ¡Increíble!

2. **Replicación asombrosa:** Las cadenas de texto también pueden multiplicarse, como replicar un hechizo para hacerlo más poderoso. ¡Imagina multiplicar la cadena "Abracadabra" por 3 y mira, obtienes "AbracadabraAbracadabraAbracadabra"! ¡Es como lanzar un hechizo múltiples veces!
3. **Clave mayúscula:** Desbloquea un potencial oculto cambiando las mayúsculas de las cadenas, como convertir "Hola" en "hOLA" u "hola" en "HOLA". Es como usar códigos secretos para desbloquear diferentes posibilidades.
4. **Búsqueda de caracteres:** Las cadenas de texto son como cofres del tesoro llenos de caracteres. En otras palabras, una cadena de texto es una secuencia de caracteres, incluidas letras (por ejemplo, "ABC...Zabc...z"), símbolos ("!@:~/>+...") o dígitos ("0123...9"), encerrados entre comillas simples o dobles (' o "). Puedes acceder a caracteres individuales dentro de una cadena usando índices, similar a descubrir coordenadas de tesoros ocultos en un mapa. Por ejemplo, al acceder al tercer carácter de la cadena 'Python', descubrirás la letra 't'. ¡Es como encontrar una joya escondida dentro de un cofre de letras!
5. **Magia transformadora:** ¿Adivina qué? Las cadenas de texto no son estáticas. Pueden transformarse mediante métodos. Puedes dividir una cadena en una lista de palabras, reemplazar palabras o caracteres específicos, o incluso unir múltiples cadenas. Es como tener una caja de herramientas mágicas para dar forma y moldear la información textual.

¡Así que deja volar tu creatividad! Usa las cadenas de texto para crear historias caprichosas, chistes divertidos o incluso tu propio lenguaje secreto. ¡Las cadenas de texto en Python son tus fieles compañeras y una poderosa herramienta para la manipulación de texto!

¡Que bien! Ahora veamos algunos ejemplos de código para ver las cadenas de texto en acción! Considera usar libros clásicos infantiles para ilustrar los elementos que mencionaste anteriormente.⁸⁷

¡Claro! Recorramos en el mundo encantador de los libros clásicos infantiles para explorar el poder de las cadenas de texto. Aquí tienes algunos ejemplos encantadores de código:

```

1 # Mezcla y combina (cadenas individuales)
2 print("\n=== Mezcla y combina (cadenas individuales) ===")
3 libro1 = "Alicia en el País de las Maravillas"
4 libro2 = "Peter Pan"
5 libros = libro1 + " y " + libro2
6 print(f"Libro 1: {libro1}")
7 print(f"Libro 2: {libro2}")
8 print(f"{libros} juntos forman un fantástico kit de lectura para niños.")

```

```

9
10 # Mezcla y combina (lista de cadenas)
11 print("\n=== Mezcla y combina (lista de cadenas) ===")
12 titulos = ["La Comunidad del Anillo", "Las Dos Torres", "El Retorno del Rey"]
13
14 # Usa str.join() en lugar de "+" para concatenar múltiples cadenas
15 coleccion = " y ".join(titulos)
16 print(f"{coleccion} son los famosos libros de la trilogía de El Señor de los Anillos.")

```

```

=== Mezcla y combina (cadenas individuales) ===
Libro 1: Alicia en el País de las Maravillas
Libro 2: Peter Pan
Alicia en el País de las Maravillas y Peter Pan juntos forman un fantástico kit de
    lectura para niños.

=== Mezcla y combina (lista de cadenas) ===
La Comunidad del Anillo y Las Dos Torres y El Retorno del Rey son los famosos libros
    de la trilogía de El Señor de los Anillos.

```

```

1 # Replicación asombrosa
2 print("\n=== Replicación asombrosa ===")
3 frase = "\tSoplaré y soplaré y tu casa derribaré...\n"
4 habla_lobo = frase * 3
5 dialogo = "Y entonces,\n" + habla_lobo + "exclamó el lobo."
6 print(dialogo)

```

```

=== Replicación asombrosa ===
Y entonces,
    Soplaré y soplaré y tu casa derribaré...
    Soplaré y soplaré y tu casa derribaré...
    Soplaré y soplaré y tu casa derribaré...
exclamó el lobo.

```

```

1 # Clave mayúscula
2 print("\n=== Clave mayúscula ===")
3 libro = "Harry Potter y la piedra filosofal"
4 estilo_titulo = libro.title()
5 solo_mayusculas = libro.upper()
6 solo_minusculas = libro.lower()
7 invertir_mayus_minus = libro.swapcase()

```

```

8 print("Texto original:\t\t", libro)
9 print("Mayúsculas y minúsculas:\t", estilo_titulo)
10 print("Todo en mayúsculas:\t", solo_mayusculas)
11 print("Todo en minúsculas:\t", solo_minusculas)
12 print("Invertir mayúsculas y minúsculas:\t", invertir_mayus_minus)

```

=== Clave mayúscula ===

```

Texto original:      Harry Potter y la piedra filosofal
Estilo título:       Harry Potter Y La Piedra Filosofal
Todo en mayúsculas:  HARRY POTTER Y LA PIEDRA FILOSOFAL
Todo en minúsculas:  harry potter y la piedra filosofal
Invertir mayúsculas y minúsculas:  hARRY pOTTER Y LA PIEDRA FILOSOFAL

```

```

1 # Búsqueda de caracteres
2 print("\n=== Búsqueda de caracteres ===")
3 cuento = "Caperucita Roja"
4 print(f"Título original:\t\t {cuento}")
5 print(f"Primer carácter:\t {cuento[0]}") # Acceder al primer carácter con índice 0
6 print(f"Primeros 6 caracteres:\t {cuento[:6]}") # Segmenta los primeros 6 caracteres
7 print(f"Cada dos caracteres:\t {cuento[::2]}") # Recorrer cada dos caracteres
8 vocales = "aeiou"
9 print(f"Solo consonantes:\t {''.join([c for c in cuento if c not in vocales])}")
10 print(f"Vocales encontradas:\t {set([c for c in cuento if c in vocales])}")

```

=== Búsqueda de caracteres ===

```

Título original:      Caperucita Roja
Primer carácter:      C
Primeros 6 caracteres: Caperu
Cada dos caracteres:   Cprct oa
Solo consonantes:     Cprct Rj
Vocales encontradas:  {'e', 'i', 'a', 'u', 'o'}

```

```

1 # Magia transformadora
2 print("\n=== Magia transformadora ===")
3 cita = "Espejito, espejito, dime quién es la más bella de todas?"
4 palabras_separadas = cita.split()
5 palabra_reemplazada = cita.replace("bella", "[malvada]")
6 espacios_eliminados = cita.replace(" ", "")
7 cadena_invertida = cita[::-1]
8 palabras_invertidas = palabras_separadas[::-1]

```

```

9 print("Cita original:\t\t", cita)
10 print("Longitud en caracteres:\t", len(cita))
11 print("Palabras separadas:\t", palabras_separadas)
12 print("Longitud en palabras:\t", len(palabras_separadas))
13 print("Palabra reemplazada:\t", palabra_reemplazada)
14 print("Espacios eliminados:\t", espacios_eliminados)
15 print("Cadena invertida:\t", cadena_invertida)
16 print("Palabras invertidas:\t", palabras_invertidas)
17 print("Completar oración:")
18 for i in range(len(palabras_separadas)):
19     print("\t" + " ".join(palabras_separadas[:i + 1]))

```

=== Magia transformadora ===

```

Cita original:      Espejito, espejito, dime quién es la más bella de todas?
Longitud en caracteres:  56
Palabras separadas:  ['Espejito,', 'espejito,', 'dime', 'quién', 'es', 'la', 'más',
                      'bella', 'de', 'todas?']
Longitud en palabras:    10
Palabra reemplazada:  Espejito, espejito, dime quién es la más [malvada] de todas?
Espacios eliminados:  Espejito,espejito,dimequiéneslamásbelladetodas?
Cadena invertida:    ?sadot ed alleb sám al se néuiq emid ,otijepse ,otijepsE
Palabras invertidas:  ['todas?', 'de', 'bella', 'más', 'la', 'es', 'quién', 'dime',
                      'espejito,', 'Espejito,']
Completar oración:
    Espejito,
    Espejito, espejito,
    Espejito, espejito, dime
    Espejito, espejito, dime quién
    Espejito, espejito, dime quién es
    Espejito, espejito, dime quién es la
    Espejito, espejito, dime quién es la más
    Espejito, espejito, dime quién es la más bella
    Espejito, espejito, dime quién es la más bella de
    Espejito, espejito, dime quién es la más bella de todas?

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/y55mdy6n>

```

1 # Más Trucos
2 print("\n=== Más Trucos ===")
3 libros = ["La Bella y la Bestia", "El Principito", "El Mago de Oz", "20000 Leguas
           de Viaje Submarino", "El Libro de la Selva"]

```



```

4 for libro in libros:
5     # Buscar
6     print("\nTítulo del libro:\t\t", libro)
7     print("'Oz' encontrado en la posición:", libro.find("Oz"))
8     # Comienza con y Termina con
9     print("Comienza con 'El':\t", libro.startswith("El"))
10    print("Termina con 'Selva':\t", libro.endswith("Selva"))
11    # EsAlfa y EsDígito
12    print("Comienza con una palabra:\t", libro[1:6].isalpha())
13    print("Comienza con un número:\t", libro[1:6].isdigit())
14    # Eliminar
15    print('Sin comillas dobles:', libro.strip(''))

```

=== Más Trucos ===

```

Título del libro:      "La Bella y la Bestia"
'Oz' encontrado en la posición: -1
Comienza con 'El':    False
Termina con 'Selva':  False
Comienza con una palabra:  True
Comienza con un número:  False
Sin comillas dobles: La Bella y la Bestia

```

```

Título del libro:      "El Principito"
'Oz' encontrado en la posición: -1
Comienza con 'El':    True
Termina con 'Selva':  False
Comienza con una palabra:  True
Comienza con un número:  False
Sin comillas dobles: El Principito

```

```

Título del libro:      "El Mago de Oz"
'Oz' encontrado en la posición: 11
Comienza con 'El':    True
Termina con 'Selva':  False
Comienza con una palabra:  True
Comienza con un número:  False
Sin comillas dobles: El Mago de Oz

```

```

Título del libro:      "20000 Leguas de Viaje Submarino"
'Oz' encontrado en la posición: -1
Comienza con 'El':    False
Termina con 'Selva':  False
Comienza con una palabra:  False
Comienza con un número:  True
Sin comillas dobles: 20000 Leguas de Viaje Submarino

```

```

Título del libro:      "El Libro de la Selva"
'Oz' encontrado en la posición: -1
Comienza con 'El':    True
Termina con 'Selva':  True
Comienza con una palabra:  True
Comienza con un número:  False
Sin comillas dobles: El Libro de la Selva

```

Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/y55mdy6n>

El código anterior realiza varias operaciones de cadenas en una lista de títulos de libros:

1. Utiliza el método `find()` para localizar la posición de la subcadena "Oz" en cada título de libro.
2. El método `startswith()` verifica si un título de libro comienza con la subcadena "El".
3. El método `endswith()` comprueba si un título de libro termina con la subcadena "Selva".
4. Al segmentar la porción inicial del título del libro, determina si está compuesto por caracteres alfabéticos utilizando `isalpha()`, o por dígitos numéricos utilizando `isdigit()`.
5. El método `strip()` elimina las comillas dobles del título del libro.

Estas operaciones proporcionan información sobre los títulos de los libros, como encontrar subcadenas específicas, verificar prefijos y sufijos, examinar la composición de caracteres de porciones y eliminar caracteres innecesarios. ¡Así que adelante y explora más ejemplos, y diviértete descubriendo el mágico y vibrante mundo de las cadenas con todas sus increíbles herramientas!

5.5 Diccionarios

*Explícame todo lo que necesito saber sobre los diccionarios en Python de una manera divertida y creativa, utilizando ejemplos lúdicos.*⁸⁸

¡Seguro! Vamos a embarcarnos en un emocionante viaje al reino de los diccionarios en Python.

Imagina que tienes un universo de superhéroes donde cada héroe posee poderes únicos, aliados y archienemigos y quieres crear una enciclopedia de ese superuniverso, capturando los fascinantes detalles de cada héroe. Cada héroe puede tener su propia página en esta enciclopedia, una biografía única llena de sus características y conexiones. ¡Aquí es donde los diccionarios resultan útiles!

1. **Pares de Clave-Valor:** Los diccionarios consisten de pares clave-valor, como si fueran códigos que revelan sus significados correspondientes. La identidad de cada héroe sirve como clave, y sus detalles, como poderes, aliados y enemigos, son los valores asociados a esa clave. Es como tener un poderoso sistema de indexación que conecta a los héroes con sus propios atributos.
2. **Búsquedas Ultrarápidas:** Los diccionarios destacan al responder preguntas como "¿Cuáles son los poderes de este héroe?". Pueden realizar búsquedas ultrarrápidas basadas en la identidad de un héroe. Es como tener una base de datos superpoderosa que recupera rápidamente los detalles de un héroe específico.
3. **Sin Duplicados:** Los diccionarios aseguran que cada héroe sea único, así como en un superuniverso no pueden existir dos héroes con la misma identidad. Si intentas agregar un héroe con una identidad existente, el diccionario mantendrá solo una instancia de ellos, preservando la singularidad de cada héroe.
4. **Dinámicos y Versátiles:** Los diccionarios son dinámicos y versátiles. Puedes modificarlos sobre la marcha, actualizar sus poderes o aliados, agregar nuevos héroes o eliminar los existentes. Proporcionan una estructura flexible para adaptarse al mundo en constante evolución de los superhéroes.
5. **Referencias Cruzadas:** Los diccionarios te permiten desentrañar las intrincadas relaciones entre héroes y villanos, y conectar a los héroes con sus aliados. Puedes buscar villanos basados en héroes o viceversa, descubriendo batallas épicas y rivalidades dentro del universo de héroes. Los diccionarios también te permiten explorar la intrincada red de relaciones entre héroes, creando una comunidad interconectada de superhéroes.

Recuerda, los diccionarios son tu puerta de entrada para construir contenedores dinámicos llenos de elementos únicos y sus atributos asociados. Te permiten crear, manipular y explorar su contenido de manera flexible y versátil. ¡Así que te animo a descubrir los secretos de los diccionarios en Python y aprovechar su poder para liberar tu creatividad!

¡Genial! ¡Ahora veamos algunos ejemplos de código para ver los diccionarios en acción! Mantengámonos en el tema de los superhéroes para ilustrar los elementos que mencionaste anteriormente.⁸⁹

¡Por supuesto! Aquí tienes algunos ejemplos de código que muestran el poder de los diccionarios en el reino de los superhéroes:

```

1 # Crear un diccionario para un superhéroe
2 spiderman = {
3     "poderes": ["superfuerza", "trepar paredes", "lanzamiento de telarañas"],
4     "aliados": ["Iron Man", "Capitán América", "Viuda Negra"],
5     "enemigos": ["Duende Verde", "Veneno", "Doctor Octopus"]
6 }
7
8 # Pares de Clave-Valor
9 print("\n=== Pares de Clave-Valor ===")
10 print("\nClaves en el diccionario de Spiderman:")
11 for clave in spiderman.keys():
12     print(f"\t{clave}")
13
14 print("\nValores en el diccionario de Spiderman:")
15 for valor in spiderman.values():
16     print(f"\t{valor}")
17
18 print("\nPares de Clave-Valor en el diccionario de Spiderman:")
19 for clave, valor in spiderman.items():
20     print(f"{clave} -> {valor}")

```

```
=== Pares de Clave-Valor ===
```

```
Claves en el diccionario de Spiderman:
```

```

    poderes
    aliados
    enemigos

```

```
Valores en el diccionario de Spiderman:
```

```

    ['superfuerza', 'trepar paredes', 'lanzamiento de telarañas']
    ['Iron Man', 'Capitán América', 'Viuda Negra']
    ['Duende Verde', 'Veneno', 'Doctor Octopus']

```

```
Pares de Clave-Valor en el diccionario de Spiderman:
```

```

poderes -> ['superfuerza', 'trepar paredes', 'lanzamiento de telarañas']
aliados -> ['Iron Man', 'Capitán América', 'Viuda Negra']
enemigos -> ['Duende Verde', 'Veneno', 'Doctor Octopus']

```

```

1 # Búsquedas Ultrarápidas
2 print("\n=== Búsquedas Ultrarápidas ===")
3 # Los atributos se indexan por clave
4 print("Poderes de Spider-Man:", spiderman["poderes"])
5 print("Aliados de Spider-Man:", spiderman["aliados"])
6 print("Enemigos de Spider-Man:", spiderman["enemigos"])

```

```

=== Búsquedas Ultrarápidas ===
Poderes de Spider-Man: ['superfuerza', 'trepar paredes', 'lanzamiento de telarañas']
Aliados de Spider-Man: ['Iron Man', 'Capitán América', 'Viuda Negra']
Enemigos de Spider-Man: ['Duende Verde', 'Veneno', 'Doctor Octopus']

```

```

1 # Sin Duplicados
2 print("\n=== Sin Duplicados ===")
3 print("Poderes de Spider-Man:", spiderman["poderes"])
4
5 # Clave duplicada, se sobrescribe
6 spiderman.update({"poderes" : ["resistencia", "disfraz vistoso"]})
7 print("Poderes actualizados de Spider-Man:", spiderman["poderes"])

```

```

=== Sin Duplicados ===
Poderes de Spider-Man: ['superfuerza', 'trepar paredes', 'lanzamiento de telarañas']
Poderes actualizados de Spider-Man: ['resistencia', 'disfraz vistoso']

```

```

1 # Dinámicos y Versátiles
2 print("\n=== Dinámicos y Versátiles ===")
3 print("Enemigos de Spider-Man:\t", spiderman["enemigos"])
4
5 spiderman["enemigos"].remove("Duende Verde")
6 spiderman["enemigos"].append("Carnage")
7 print("Enemigos actualizados de Spider-Man:", spiderman["enemigos"])

```

```

=== Dinámicos y Versátiles ===
Enemigos de Spider-Man: ['Duende Verde', 'Veneno', 'Doctor Octopus']
Enemigos actualizados de Spider-Man: ['Veneno', 'Doctor Octopus', 'Carnage']

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/epvbefcm>

¿Qué tal crear una enciclopedia de superhéroes? ¿Puedo implementarla como un diccionario de diccionarios? ¿Alguna especie de diccionario anidado? Ilústralo con un ejemplo de código.⁹⁰

¡Por supuesto! Puedes crear una enciclopedia de superhéroes utilizando una estructura de diccionario anidado. Cada superhéroe puede ser una clave en el diccionario externo, y el valor correspondiente puede ser un diccionario interno que almacene los detalles del superhéroe. Aquí tienes un ejemplo de código junto con las salidas correspondientes:

```

1 # Crea la enciclopedia de superhéroes
2
3 superheroes = {
4     "Mujer Maravilla": {
5         "poderes": ["Superfuerza", "Lazo de la Verdad", "Vuelo"],
6         "aliados": ["Superman", "Batman"],
7         "enemigos": ["Cheetah", "Ares"],
8         "debut": 1941
9     },
10    "Viuda Negra": {
11        "poderes": ["Espía maestra", "Combate cuerpo a cuerpo", "Espionaje"],
12        "aliados": ["Capitán América", "Ojo de Halcón"],
13        "enemigos": ["Taskmaster", "Guardián Rojo"],
14        "debut": 1964
15    },
16    "Capitana Marvel": {
17        "poderes": ["Superfuerza", "Proyección de energía", "Vuelo"],
18        "aliados": ["Iron Man", "Thor"],
19        "enemigos": ["Thanos", "Yon-Rogg"],
20        "debut": 1968
21    },
22    "Tormenta": {
23        "poderes": ["Manipulación del clima", "Vuelo", "Absorción de energía"],
24        "aliados": ["Wolverine", "Cíclope"],
25        "enemigos": ["Magneto", "Apocalipsis"],
26        "debut": 1975
27    },
28    "Superman": {
29        "poderes": ["Superfuerza", "Vuelo", "Visión de calor"],
30        "aliados": ["Batman", "Mujer Maravilla"],
31        "enemigos": ["Lex Luthor", "Doomsday"],
32        "debut": 1938
33    },

```

```

34 "Spider-Man": {
35     "poderes": ["Tregar paredes", "Sentido arácnido", "Lanzamiento de telarañas"],
36     "aliados": ["Iron Man", "Capitán América"],
37     "enemigos": ["Duende Verde", "Veneno", "Doctor Octopus"],
38     "debut": 1962
39 },
40 "Hulk": {
41     "poderes": ["Superfuerza", "Invulnerabilidad", "Regeneración"],
42     "aliados": ["Thor", "Viuda Negra"],
43     "enemigos": ["Abominación", "Hulk Rojo"],
44     "debut": 1962
45 },
46 "Batman": {
47     "poderes": ["Inteligencia", "Riqueza", "Artes marciales"],
48     "aliados": ["Superman", "Mujer Maravilla"],
49     "enemigos": ["Joker", "Bane"],
50     "debut": 1939
51 },
52 "Wolverine": {
53     "poderes": ["Regeneración", "Garras de adamantium", "Sentidos agudizados"],
54     "aliados": ["Profesor X", "Cíclope", "Tormenta"],
55     "enemigos": ["Dientes de Sable", "Magneto", "Mística"],
56     "debut": 1974
57 }
58 }
59 print(superheroes.keys())

```

```

dict_keys(['Mujer Maravilla', 'Viuda Negra', 'Capitana Marvel', 'Tormenta',
'Superman', 'Spider-Man', 'Hulk', 'Batman', 'Wolverine'])

```

```

1 # Imprime detalles de un superhéroe dado su nombre en la enciclopedia
2 nombre = input("\nIngresa un nombre para buscar en la enciclopedia de superhéroes: ")
3 nombre = nombre.title() # Capitalizar primera letra de cada palabra en el nombre
4
5 if nombre in superheroes:
6     superheroe = superheroes[nombre]
7     print("-----")
8     print(f"{nombre:^20}")
9     print("-----")
10    print(f"Poderes: {'', '.join(superheroe['poderes'])}")
11    print(f"Aliados: {'', '.join(superheroe['aliados'])}")

```

```

12     print(f"Enemigos: {' '.join(superheroe['enemigos'])}")
13     print(f"Año de debut: {superheroe['debut']}")
14 else:
15     print("Lo siento, superhéroe no encontrado en la enciclopedia.")

```

Ingresa un nombre para buscar en la enciclopedia de superhéroes: Thor
Lo siento, superhéroe no encontrado en la enciclopedia.

Ingresa un nombre para buscar en la enciclopedia de superhéroes: viuda negra

Viuda Negra

Poderes: Espía maestra, Combate cuerpo a cuerpo, Espionaje
Aliados: Capitán América, Ojo de Halcón
Enemigos: Taskmaster, Guardián Rojo
Año de debut: 1964

```

1 # Agrega un nuevo superhéroe
2 superheroes["Capitán América"] = {
3     "poderes": ["Superfuerza", "Lanzamiento de escudo", "Liderazgo"],
4     "aliados": ["Iron Man", "Thor"],
5     "enemigos": ["Calavera Roja", "Soldado del Invierno"],
6     "debut": 1941
7 }
8 # Elimina un superhéroe
9 del superheroes["Hulk"]
10 # Actualiza detalles de superhéroes
11 superheroes["Wolverine"]["aliados"].append("Jean Grey")
12 superheroes["Spider-Man"]["enemigos"].remove("Doctor Octopus")
13 # Imprime la enciclopedia de superhéroes actualizada
14 valor = input("\n¿Qué atributo (poderes/aliados/enemigos/debut) te gustaría ver? ")
15 valor = valor.lower() # Convertir el atributo a minúsculas
16 print("-----")
17 print(f"{'Enciclopedia de Superhéroes':^50}")
18 print("-----")
19
20 for clave in superheroes.keys():
21     if valor in superheroes[clave]:
22         print(f"{clave:^20} -> {superheroes[clave][valor]}")
23     else:
24         print(f"Lo siento, no se encontró {valor} para el superhéroe {clave}")

```


¿Qué atributo (poderes/aliados/enemigos/debut) te gustaría ver? aliados

Enciclopedia de Superhéroes

```
Mujer Maravilla  -> ['Superman', 'Batman']
Viuda Negra      -> ['Capitán América', 'Ojo de Halcón']
Capitana Marvel  -> ['Iron Man', 'Thor']
Tormenta         -> ['Wolverine', 'Cíclope']
Superman         -> ['Batman', 'Mujer Maravilla']
Spider-Man       -> ['Iron Man', 'Capitán América']
Batman           -> ['Superman', 'Mujer Maravilla']
Wolverine        -> ['Profesor X', 'Cíclope', 'Tormenta', 'Jean Grey']
Capitán América  -> ['Iron Man', 'Thor']
```

¿Qué atributo (poderes/aliados/enemigos/debut) te gustaría ver? edad

Enciclopedia de Superhéroes

```
Lo siento, no se encontró edad para el superhéroe Mujer Maravilla
Lo siento, no se encontró edad para el superhéroe Viuda Negra
Lo siento, no se encontró edad para el superhéroe Capitana Marvel
Lo siento, no se encontró edad para el superhéroe Tormenta
Lo siento, no se encontró edad para el superhéroe Superman
Lo siento, no se encontró edad para el superhéroe Spider-Man
Lo siento, no se encontró edad para el superhéroe Batman
Lo siento, no se encontró edad para el superhéroe Wolverine
```

¿Qué atributo (poderes/aliados/enemigos/debut) te gustaría ver? debut

Enciclopedia de Superhéroes

```
Mujer Maravilla  -> 1941
Viuda Negra      -> 1964
Capitana Marvel  -> 1968
Tormenta         -> 1975
Superman         -> 1938
Spider-Man       -> 1962
Batman           -> 1939
Wolverine        -> 1974
Capitán América  -> 1941
```

↙ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/5feexkfe>

¡Súper! También mencionaste que los diccionarios nos permiten explorar relaciones y conexiones complejas entre los elementos dentro del diccionario. ¿Qué significa eso? ¿Pueden los diccionarios ayudarme a determinar relaciones como “los enemigos de mis aliados son mis enemigos”, “los aliados de mis aliados son mis amigos”, “los héroes con el mismo superpoder que el mío son mis amigos” o “mostrar una tabla de superhéroes ordenados por año de estreno”? Por favor, proporciona algunos ejemplos.⁹¹

¡Claro que sí! Aquí tienes cuatro ejemplos de cómo se pueden cruzar y relacionar datos utilizando diccionarios.

► Ejemplo 1: Los enemigos de mis aliados son mis enemigos.

```

1 # Construye un diccionario para almacenar enemigos extendidos
2 aliados_enemigos = {}
3 for hero, detalles in superheroes.items():
4     aliados = detalles["aliados"]
5     enemigos = detalles["enemigos"]
6     for aliado in [hero] + aliados:
7         # Comprueba si el héroe aún no está presente en aliados_enemigos
8         if hero not in aliados_enemigos:
9             aliados_enemigos[hero] = []
10        # Comprueba si el aliado existe en el diccionario superheroes
11        if aliado in superheroes:
12            # Ampliar la lista de enemigos del héroe con los enemigos del aliado
13            aliados_enemigos[hero].extend(superheroes[aliado]["enemigos"])
14
15 # Imprime los enemigos originales
16 print('-'*80)
17 print(f'{"Enemigos Originales":^80}')
18 print('-'*80)
19 for hero, detalles in superheroes.items():
20     print(f"{hero:^10} -> {detalles['enemigos']}")
21
22 # Imprime los enemigos extendidos
23 print('-'*80)
24 print(f'{"Los Enemigos de Mis Aliados Son Mis Enemigos":^80}')
25 print('-'*80)
26 for aliado, enemigos in aliados_enemigos.items():
27     print(f"{aliado:^10} -> {enemigos}")
28

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/yckcbr4k>

El código tiene como objetivo construir un diccionario, `aliados_enemigos`, para almacenar los enemigos extendidos de los héroes en función de sus aliados. El algoritmo itera a través de cada héroe y sus detalles en el diccionario `superheroes`. Para cada héroe, recupera las listas de aliados y enemigos. Para cada héroe y sus aliados, comprueba si el héroe aún no está presente en el diccionario `aliados_enemigos`. Si no lo está, inicializa una lista vacía para el héroe. Luego, comprueba si el aliado existe en el diccionario `superheroes`. Si el aliado existe, extiende la lista de enemigos del héroe con los enemigos del aliado utilizando la función `extend()`.

El resultado final del código muestra los enemigos originales y los enemigos extendidos (en función de los aliados) mediante la impresión de los diccionarios `superheroes` y `aliados_enemigos`.

```
-----
                        Enemigos Originales
-----
Mujer Maravilla -> ['Cheetah', 'Ares']
Viuda Negra    -> ['Taskmaster', 'Guardián Rojo']
Capitana Marvel -> ['Thanos', 'Yon-Rogg']
Tormenta       -> ['Magneto', 'Apocalipsis']
Superman       -> ['Lex Luthor', 'Doomsday']
Spider-Man     -> ['Duende Verde', 'Veneno']
Batman         -> ['Joker', 'Bane']
Wolverine      -> ['Dientes de Sable', 'Magneto', 'Mística']
Capitán América -> ['Calavera Roja', 'Soldado del Invierno']

-----
                        Los Enemigos de Mis Aliados Son Mis Enemigos
-----
Mujer Maravilla -> ['Cheetah', 'Ares', 'Lex Luthor', 'Doomsday', 'Joker',
                    'Bane']
Viuda Negra     -> ['Taskmaster', 'Guardián Rojo', 'Calavera Roja', 'Soldado
                    del Invierno']
Capitana Marvel -> ['Thanos', 'Yon-Rogg']
Tormenta        -> ['Magneto', 'Apocalipsis', 'Dientes de Sable', 'Magneto',
                    'Mística']
Superman        -> ['Lex Luthor', 'Doomsday', 'Joker', 'Bane', 'Cheetah',
                    'Ares']
Spider-Man      -> ['Duende Verde', 'Veneno', 'Calavera Roja', 'Soldado del
                    Invierno']
Batman          -> ['Joker', 'Bane', 'Lex Luthor', 'Doomsday', 'Cheetah',
                    'Ares']
```

```

Wolverine    -> ['Dientes de Sable', 'Magneto', 'Mística', 'Magneto',
                'Apocalipsis']
Capitán América -> ['Calavera Roja', 'Soldado del Invierno']

```

► Ejemplo 2: Los aliados de mis aliados son mis amigos.

```

1 # Construye un diccionario para almacenar amigos extendidos
2 aliados_amigos = {}
3 for hero, detalles in superheroes.items():
4     aliados = detalles["aliados"]
5     for aliado in [hero] + aliados:
6         if aliado in superheroes:
7             # Encuentra amigos del aliado actual, excluyendo al héroe mismo
8             amigos = set(superheroes[aliado]["aliados"]) - set([hero])
9             aliados_amigos.setdefault(hero, []).extend(amigos)
10
11 # Imprime los aliados originales
12 print('-'*80)
13 print(f"{'Aliados Originales':^80}")
14 print('-'*80)
15 for hero, detalles in superheroes.items():
16     # Imprime los aliados de cada héroe
17     print(f"{hero:^10} -> {detalles['aliados']}")
18
19 # Imprime los aliados extendidos (amigos)
20 print('-'*80)
21 print(f"{'Los Aliados de Mis Aliados Son Mis Amigos':^80}")
22 print('-'*80)
23 for hero, amigos in aliados_amigos.items():
24     # Imprimir la lista extendida de amigos, excluyendo duplicados
25     print(f"{hero:^10} -> {set(amigos)}")
26

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/4wsy98ra>

El código tiene como objetivo construir un diccionario, `aliados_amigos`, para almacenar los amigos extendidos de cada héroe en función de sus aliados. El algoritmo itera a través de cada héroe y sus detalles en el diccionario `superheroes`. Para cada héroe, recupera su lista de aliados. Luego, itera a través de cada aliado y usa la función `append()` para agregar el héroe a la lista de amigos asociada con ese aliado en el diccionario `aliados_amigos`.

La función `setdefault()` asegura que, si el héroe aún no es una clave en el diccionario `aliados_amigos`, se crea una nueva lista vacía como valor predeterminado. Esto permite

ampliar fácilmente la lista de amigos sin sobrescribir los amigos existentes. Finalmente, el código imprime los aliados originales de cada héroe y los aliados extendidos (con amigos) mediante la iteración del diccionario `aliados_amigos` y excluyendo duplicados utilizando conjuntos.

```
-----
                        Aliados Originales
-----
Mujer Maravilla -> ['Superman', 'Batman']
Viuda Negra -> ['Capitán América', 'Ojo de Halcón']
Capitana Marvel -> ['Iron Man', 'Thor']
    Tormenta    -> ['Wolverine', 'Cíclope']
    Superman    -> ['Batman', 'Mujer Maravilla']
Spider-Man -> ['Iron Man', 'Capitán América']
    Batman      -> ['Superman', 'Mujer Maravilla']
Wolverine -> ['Professor X', 'Cíclope', 'Tormenta', 'Jean Grey']
Capitán América -> ['Iron Man', 'Thor']

-----
                        Los Aliados de Mis Aliados Son Mis Amigos
-----
Mujer Maravilla -> {'Superman', 'Batman'}
Viuda Negra -> {'Iron Man', 'Capitán América', 'Ojo de Halcón', 'Thor'}
Capitana Marvel -> {'Iron Man', 'Thor'}
    Tormenta    -> {'Cíclope', 'Professor X', 'Wolverine', 'Jean Grey'}
    Superman    -> {'Batman', 'Mujer Maravilla'}
Spider-Man -> {'Capitán América', 'Iron Man', 'Thor'}
    Batman      -> {'Superman', 'Mujer Maravilla'}
Wolverine -> {'Tormenta', 'Professor X', 'Cíclope', 'Jean Grey'}
Capitán América -> {'Iron Man', 'Thor'}
```

► Ejemplo 3: Héroes con los mismos superpoderes son amigos.

```
1 # Construye un diccionario para almacenar héroes amigos según superpoderes
2 superpoder_amigos = {}
3 for hero, detalles in superheroes.items():
4     poderes = detalles["poderes"]
5     for poder in poderes:
6         superpoder_amigos.setdefault(poder, []).append(hero)
7 print('-'*80)
8 print(f"{'Héroes con los Mismos Superpoderes Son Amigos':^80}")
9 print('-'*80)
```

```

10 for poder, amigos in superpoder_amigos.items():
11     print(f"|{poder.title():^20}| {'| '}.join(amigos):<56}|")
12     print('-'*80)
13

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/4mvsc3s>

El código tiene como objetivo construir un diccionario, `superpoder_amigos`, para almacenar los superhéroes que comparten los mismos superpoderes. El algoritmo itera a través de cada héroe y sus detalles en el diccionario `superheroes`. Para cada héroe, recupera su lista de poderes. Luego, itera a través de cada poder y utiliza la función `append()` para agregar el héroe a la lista de amigos asociada con ese poder en el diccionario `superpoder_amigos`.

La función `setdefault()` asegura que, si el poder aún no es una clave en el diccionario `superpoder_amigos`, se crea una nueva lista vacía como valor predeterminado. Esto permite agregar fácilmente héroes a la lista de amigos sin sobrescribir héroes existentes. Finalmente, el código imprime el diccionario `superpoder_amigos`, mostrando cada poder y los amigos correspondientes. Formatea la salida en una estructura similar a una tabla utilizando el carácter `|` para separar las columnas de poder y amigos.

```

-----
                        Héroes con los Mismos Superpoderes Son Amigos
-----
| Super Fuerza | Mujer Maravilla, Capitana Marvel, Superman, Capitán América|
-----
| Lazo de la Verdad | Mujer Maravilla                                     |
-----
|      Vuelo      | Mujer Maravilla, Capitana Marvel, Tormenta, Superman      |
-----
| Espía Maestro  | Viuda Negra                                     |
-----
| Combate Cuerpo a Cuerpo | Viuda Negra                                     |
-----
| Espionaje       | Viuda Negra                                     |
-----
| Proyección de Energía | Capitana Marvel                                     |
-----
| Manipulación del Clima | Tormenta                                           |
-----
| Absorción de Energía | Tormenta                                           |
-----
| Visión de Calor  | Superman                                           |
-----

```

	Escalada de Paredes	Spider-Man	

	Sentido Arácnido	Spider-Man	

	Lanzamiento de Telaraña	Spider-Man	

	Inteligencia	Batman	

	Riqueza	Batman	

	Artes Marciales	Batman	

	Regeneración	Wolverine	

	Garras de Adamantium	Wolverine	

	Sentidos Agudizados	Wolverine	

	Lanzamiento de Escudo	Capitán América	

	Liderazgo	Capitán América	

► Ejemplo 4: Ordenar superhéroes por año de debut.

```

1 # Crea una lista ordenada de claves basadas en el año de debut
2 debut_dict = {clave: superheroes[clave]["debut"] for clave in superheroes.keys()}
3 sorted_keys = sorted(debut_dict.keys(), key=debut_dict.get)
4
5 # Imprime la tabla de superhéroes ordenados por año de debut
6 print('-'*30)
7 print(f'{'Superhéroes Ordenados por Año de Debut':^30}')
8 print('-'*30)
9 print(f'{'Superhéroe':^15} | {'Año de Debut':^15}')
10 print('-'*30)
11 for clave in sorted_keys:
12     print(f"{clave:<15} | {superheroes[clave]['debut']:^15}")
13 print('-'*30)
14

```

↙ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/dshc9uf4>

Este código primero crea un diccionario `debut_dict` para almacenar los años de debut de los superhéroes. La definición por comprensión del diccionario `{clave : superheroes[clave]["debut"] for clave in superheroes.keys()}` itera sobre las claves del diccionario `superheroes` y asigna el año de debut correspondiente a cada clave en `debut_dict`. Este paso extrae efectivamente los años de debut para cada superhéroe, similar a la definición por comprensión de listas que vimos anteriormente.

La segunda línea del código ordena las claves en `debut_dict` basándose en sus valores correspondientes (años de debut) utilizando la función `sorted()`. El parámetro `key` se establece en `debut_dict.get`, que recupera el valor (año de debut) para cada clave. Este paso de ordenación asegura que los superhéroes estén dispuestos en orden ascendente según su año de debut. El último bloque del código imprime la tabla mediante la iteración a través de la lista ordenada de claves y aplicando algunas opciones de formato.

```
-----
Superhéroes Ordenados por Año de Debut
-----
Superhéroe | Año de Debut
-----
Superman   | 1938
Batman     | 1939
Mujer Maravilla | 1941
Capitán América | 1941
Spider-Man  | 1962
Viuda Negra | 1964
Capitana Marvel | 1968
Wolverine   | 1974
Tormenta    | 1975
-----
```

¡Vamos! Deja volar tu imaginación mientras exploras más ejemplos y disfrutas descubriendo el fascinante mundo de los diccionarios con todas sus increíbles características y posibilidades. ¡Prepárate para experimentar la magia de la organización y recuperación de datos!

5.6 ¡Manos a la obra!

Sugiere algunos ejercicios divertidos e interesantes para practicar con listas, tuplas y conjuntos. Proporciona una descripción del problema y ejemplos de los resultados esperados.⁹²

1. **Manipulación de listas:** Dada una lista de números, escribe un programa que realice las siguientes operaciones:

- ▶ Eliminar todos los números pares de la lista.
- ▶ Elevar al cuadrado cada número restante de la lista.
- ▶ Imprime la lista actualizada.

Ejemplo:

```
Entrada: [1, 2, 3, 4, 5, 6, 7, 8]
Salida: [1, 9, 25, 49]
```

2. **Concatenación de tuplas:** Dadas dos tuplas, escribe un programa que las concatene e imprima la nueva tupla. Ejemplo:

```
Entrada:
Tupla1: (1, 2, 3)
Tupla2: (4, 5, 6)
Salida: (1, 2, 3, 4, 5, 6)
```

3. **Operaciones de conjuntos con cadenas:** Dados dos conjuntos de cadenas, realiza las siguientes operaciones e imprime los resultados:

- ▶ Unión: Encuentra la unión de los dos conjuntos.
- ▶ Intersección: Encuentra las cadenas comunes entre los dos conjuntos.
- ▶ Diferencia: Encuentra las cadenas que están en el primer conjunto pero no en el segundo conjunto.

Ejemplo:

```
Entrada:
Conjunto1: {"manzana", "banano", "naranja"}
Conjunto2: {"naranja", "uva", "kiwi"}
Salida:
Unión: {"manzana", "banano", "naranja", "uva", "kiwi"}
Intersección: {"naranja"}
Diferencia: {"manzana", "banano"}
```

4. **Filtrado de lista:** Dada una lista de números, escribe un programa para filtrar todos los números primos mayores que 10 e imprima la lista resultante. Ejemplo:

```
Entrada: [5, 12, 8, 17, 3, 20, 41]
Salida: [5, 12, 8, 3, 20]
```

5. **Permutaciones únicas:** Dada una lista de números, escribe un programa para generar todas las permutaciones únicas de los números. Ejemplo:

```
Entrada: [1, 2, 3]
Salida: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

6. **Ordenamiento de tuplas:** Dada una lista de tuplas que contienen nombres y edades, escribe un programa para ordenar las tuplas en función de las edades en orden descendente. Ejemplo:

```
Entrada: [("Juan", 25), ("Ana", 30), ("Carlos", 22)]
Salida: [("Ana", 30), ("Juan", 25), ("Carlos", 22)]
```

7. **Operaciones de conjuntos con múltiples conjuntos:** Dada una lista de conjuntos, escribe un programa para realizar las siguientes operaciones e imprimir los resultados:

- ▶ Unión: Encuentra la unión de todos los conjuntos.
- ▶ Intersección: Encuentra los elementos comunes en todos los conjuntos.
- ▶ Diferencia: Encuentra los elementos que están presentes en el primer conjunto pero no en los demás conjuntos.

Ejemplo:

```
Entrada:
Conjuntos: [{1, 2, 3}, {2, 3, 4}, {3, 4, 5}]
Salida:
Unión: {1, 2, 3, 4, 5}
Intersección: {3}
Diferencia: {1}
```

8. **Reorganización de lista:** Dada una lista de cadenas, escribe un programa para reorganizar las cadenas de tal manera que cada cadena comience con el último carácter de la cadena anterior. Si no hay tal cadena, continúa con cualquier cadena no utilizada de la lista. Ejemplo:

```
Entrada: ["limón", "aceituna", "nuez", "halcón", "orca", "zorro"]
Salida: ["limón", "nuez", "zorro", "orca", "aceituna", "halcón"]
```

9. **Combinaciones únicas:** Dada una lista de números, escribe un programa para generar todas las combinaciones únicas de longitud específica *k* de la lista. Ejemplo:

```
Entrada: [1, 2, 3], k = 2
Salida: [[1, 2], [1, 3], [2, 3]]
```

10. **Manipulación de tuplas:** Dada una lista de tuplas que contienen nombres y puntajes, escribe un programa para encontrar el puntaje promedio para cada nombre e imprima una lista de tuplas en el formato (**nombre**, **puntaje_promedio**), ordenadas por puntaje promedio en orden descendente. Ejemplo:

```
Entrada: [("Juan", 80), ("Ana", 90), ("Carlos", 75), ("Ana", 95)]
Salida: [("Ana", 92.5), ("Juan", 80), ("Carlos", 75)]
```

11. **Segmentación de lista:** Dada una lista de números, escribe un programa para segmentar la lista en sublistas de manera que la suma de cada sublista sea menor o igual a un valor máximo dado **k**. Las sublistas deben estar lo más equilibradas posible. Ejemplo:

```
Entrada: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k = 17
Salida: [[1, 2, 3, 4, 5], [6, 7], [8, 9], [10]]
```

12. **Operaciones con matrices:** Dadas dos matrices representadas como listas de listas, escribe un programa para realizar operaciones de matriz, como suma, resta y multiplicación. Ejemplo:

```
Entrada:
Matriz A: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Matriz B: [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
Salida:
Suma: [[10, 10, 10], [10, 10, 10], [10, 10, 10]]
Resta: [[-8, -6, -4], [-2, 0, 2], [4, 6, 8]]
Multiplicación: [[30, 24, 18], [84, 69, 54], [138, 114, 90]]
```

Sugiere algunos ejercicios divertidos e interesantes para practicar con cadenas y diccionarios, incluidos diccionarios anidados. Proporciona una descripción del problema y ejemplos de los resultados esperados.⁹³

1. **Contador de palabras:** Escribe un programa que tome una cadena como entrada y cuente la frecuencia de cada palabra en la cadena. Ejemplo:

```
Entrada: "Me encanta programar, programar y programar."
Salida: {'Me': 1, 'encanta': 1, 'programar': 3, 'y': 1}
```

2. **Mezclador de diccionarios:** Escribe un programa que tome dos diccionarios como entrada y los combine en un solo diccionario. Si una clave está presente en ambos diccionarios, los valores deben concatenarse en una lista. Ejemplo:

```

Entrada:
dict1 = {'A': 1, 'B': 2, 'C': 3}
dict2 = {'B': 4, 'D': 5}
Salida: {'A': 1, 'B': [2, 4], 'C': 3, 'D': 5}

```

3. **Acceso a diccionarios anidados:** Escribe un programa que acceda a los valores de un diccionario anidado utilizando una lista de claves. Ejemplo:

```

Entrada:
data = {'persona': {'nombre': 'Juan', 'edad': 25, 'dirección': {'ciudad':
    'Nueva York', 'país': 'EE. UU.'}}}
claves = ['persona', 'dirección', 'ciudad']
Salida: 'Nueva York'

```

4. **Contador de caracteres:** Escribe un programa que tome una cadena como entrada y cuente la frecuencia de cada carácter en la cadena. Ejemplo:

```

Entrada: "¡Hola, Mundo!"
Salida: {'i': 1, 'H': 1, 'o': 2, 'l': 1, 'a': 1, ',': 1, ' ': 1, 'M': 1, 'u':
    1, 'n': 1, 'd': 1, '!': 1}

```

5. **Manipulación de diccionarios:** Escribe un programa que realice varias manipulaciones en un diccionario. El programa debe mostrar un menú de opciones para que el usuario elija (agregar, actualizar, eliminar, existe clave?). Ejemplo:

```

Entrada:
mi_diccionario = {'A': 1, 'B': 2}
Salida:
{'A': 1, 'B': 2, 'C': 3} # Después de agregar {'C': 3}
{'A': 1, 'B': 5, 'C': 3} # Después de actualizar 'B' a 5
{'A': 1, 'C': 3} # Después de eliminar 'B'
Clave 'A' existe en el diccionario: True
Clave 'B' existe en el diccionario: False

```

6. **Comprobador de anagramas:** Escribe un programa que tome dos cadenas como entrada y determine si son anagramas entre sí. Un anagrama es una palabra o frase formada al reorganizar las letras de otra palabra o frase. Ejemplo:

```

Entrada: "lapicero", "copiarle"
Salida: True

```

7. **Reemplazo de caracteres:** Escribe un programa que tome una cadena y reemplace todas las ocurrencias de un carácter especificado por otro carácter. Ejemplo:

Entrada: "¡Hola, Mundo!", reemplazar 'o' con 'y'
 Salida: "¡Hyla, Mundy!"

8. **Inversión de diccionarios:** Escribe un programa que tome un diccionario como entrada y cree un nuevo diccionario donde las claves y los valores estén intercambiados. Ejemplo:

Entrada: {'A': 1, 'B': 2, 'C': 3}
 Salida: {1: 'A', 2: 'B', 3: 'C'}

9. **Contador de vocales:** Escribe un programa que tome una cadena como entrada y cuente el número de vocales (a, e, i, o, u) en la cadena. Ejemplo:

Entrada: "¡Hola, Mundo!"
 Salida: 4

10. **Actualización de diccionarios anidados:** Escribe un programa que actualice los valores de un diccionario anidado en función de una condición especificada. Ejemplo:

Entrada:
 data = {'A': {'puntaje': 80}, 'B': {'puntaje': 90}, 'C': {'puntaje': 70}}
 umbral = 75
 Salida: {'A': {'puntaje': 'Aprobado'}, 'B': {'puntaje': 'Aprobado'}, 'C': {'puntaje': 'Reprobado'}}

11. **Búsqueda de diccionarios anidados:** Escribe un programa que busque un valor específico en un diccionario anidado y devuelva la(s) clave(s) correspondientes donde se encuentra el valor. Ejemplo:

Entrada:
 data = {'A': {'puntaje': 80}, 'B': {'puntaje': 90}, 'C': {'puntaje': 80}, 'D': {'puntaje': 70}}
 objetivo = 80
 Salida: ['A', 'C']

12. **Eliminación de la palabra de moda:** Escribe un programa que tome una cadena como entrada y elimine la(s) palabra(s) con el mayor número de apariciones. Si varias palabras tienen el mismo número máximo de apariciones, elimina todas ellas. Ejemplo:

Entrada: "El rápido zorro marrón salta sobre el perro perezoso. El perro ladra fuerte."
 Salida: "rápido zorro marrón salta sobre el perezoso. ladra fuerte."

6.1 Modularización y reusabilidad

Explica brevemente el concepto de modularización y reusabilidad, así como por qué son útiles en proyectos de programación. Luego, utilizando analogías del mundo real, explica los diversos enfoques para implementar estas ideas de manera clara y amigable.⁹⁴

La reusabilidad y la modularización son como técnicas avanzadas en el desarrollo de software, especialmente para proyectos grandes y complejos. Vienen con una serie de beneficios que facilitan la vida de los programadores.

Cuando escribes código reutilizable, no tienes que empezar desde cero cada vez. Puedes ahorrar mucho tiempo y esfuerzo utilizando soluciones existentes en lugar de reinventar la rueda. Y con la modularización, descompones tu código en piezas más pequeñas y manejables. Es como tener piezas de un rompecabezas que encajan perfectamente. Esto hace que sea más fácil de entender, mantener y corregir errores que surjan. Además, puedes reutilizar esas piezas modulares en diferentes partes de tu proyecto.

Así que, cuando se trata de abordar esos proyectos más grandes, adoptar la reusabilidad y la modularización es clave. Te ayuda a trabajar de manera más inteligente, colaborar sin problemas y mantener tu código robusto a largo plazo. En Python, tienes un montón de técnicas y prácticas interesantes para poner en marcha estas ideas:

- 1. Funciones:** Las funciones son como herramientas especializadas en una cocina. Encapsulan tareas específicas y pueden reutilizarse en diferentes recetas. De manera similar, en la programación, las funciones encapsulan operaciones específicas para solucionar tareas puntuales y detalladas. Al utilizar funciones, los programadores pueden escribir código modular que puede reutilizarse fácilmente, lo que facilita el manejo de proyectos más grandes y complejos.
- 2. Clases y Programación Orientada a Objetos (POO):** Las clases en POO son como trabajadores especializados en una organización. Cada trabajador tiene un rol y experiencia específicos. De manera similar, las clases encapsulan datos y comportamientos en objetos, promoviendo la especialización del código. Al aprovechar las clases y objetos, los programadores pueden crear un código modular y escalable, lo que facilita el manejo de la complejidad en proyectos más grandes.
- 3. Módulos y Paquetes:** Los módulos y paquetes son como cajas de herramientas con compartimentos etiquetados para diferentes herramientas. Proporcionan una forma de organizar y empaquetar funciones y clases relacionadas, promoviendo la reutilización del código. Al usar módulos y paquetes, los programadores pueden

encontrar y reutilizar fácilmente funcionalidades del código, lo que permite el desarrollo y mantenimiento eficientes de proyectos más grandes.

4. **Librerías y Marcos de Trabajo:** Las librerías y marcos de trabajo son como libros en una biblioteca. En lugar de escribir todo desde cero, puedes usar libros de referencia. Así mismo, los programadores pueden utilizar librerías y marcos de trabajo existentes que ofrecen código y funcionalidades preconstruidos para tareas específicas. Al aprovechar estos recursos, los programadores pueden ahorrar tiempo y esfuerzo, acelerando el proceso de desarrollo de proyectos de mayor alcance.
5. **Patrones de Diseño:** Los patrones de diseño son como planos arquitectónicos para construir edificios. Proporcionan soluciones probadas para problemas comunes en el diseño de software. Al seguir patrones de diseño establecidos, los programadores pueden escribir un código modular y escalable que aborda requisitos complejos. Los patrones de diseño promueven la reutilización y mantenibilidad del código, lo que los convierte en herramientas fundamentales para proyectos de alta complejidad.
6. **Separación de Responsabilidades:** La separación de responsabilidades es como dividir un proyecto entre expertos con áreas de especialización específicas. Al descomponer un programa en módulos más pequeños, cada uno responsable de una tarea o funcionalidad específica, los programadores pueden lograr la modularidad y reusabilidad del código. Este enfoque permite una mejor organización, depuración y mantenimiento del código en proyectos de mayor envergadura.

Al emplear estas técnicas y prácticas, los desarrolladores de Python pueden crear un código reutilizable y modular que es más fácil de entender, mantener y extender. Estos conceptos también contribuyen a la calidad general del código, la colaboración y la escalabilidad.

¡Uff! Que impresionante lista de ideas poderosas. Sin embargo, como aún estoy aprendiendo, por favor, recomiéndame uno o dos elementos de la lista con los que debería comenzar como principiante, y luego describe los componentes clave de cada elemento de manera relajada y entretenida.⁹⁵

¡Sin problema! Como principiante, te recomiendo comenzar con funciones y clases. Estos conceptos son bloques de construcción fundamentales en la programación y te proporcionarán una base sólida para comprender tópicos más avanzados.

1. **Funciones:** Las funciones son como recetas gourmet en el mundo de la programación. Así como una receta te guía para crear un delicioso plato, las funciones guían a la computadora para realizar tareas concretas. Piensa en las funciones como mini-programas dentro de tu programa. Tienen un nombre y un conjunto de instrucciones que realizan una acción particular. Las funciones pueden recibir entradas (ingredientes) y producir salidas (el plato final). ¿Lo mejor? Una vez que creas una función, puedes usarla una y otra vez, ahorrándote tiempo y esfuerzo. Aquí tienes algunos detalles técnicos más sobre las funciones en Python:

- ▶ **Solucionadores Especializados:** Las funciones son como solucionadores especializados de problemas. Reciben entradas (parámetros) y realizan operaciones específicas en ellas para producir salidas (valores de retorno). Las funciones ayudan a descomponer la complejidad de los problemas en piezas manejables. Son como te dije, mini-programas dentro de tu código que se pueden llamar y reutilizar siempre que las necesites. Las funciones se definen utilizando la instrucción `def`.
 - ▶ **Valores de Retorno:** Las funciones pueden devolver valores utilizando la instrucción `return`. Esto les permite producir un resultado o salida que se puede usar en otras partes del código. Es como si una función te devolviera un regalo después de realizar su tarea.
 - ▶ **Parámetros:** Las funciones pueden aceptar parámetros, que son entradas proporcionadas cuando se llama a la función. Los parámetros te permiten pasar valores a la función, proporcionando los datos necesarios para que la función funcione (valga la redundancia). Es como darle instrucciones o información a la función para que puedan realizar su trabajo.
 - ▶ **Funciones Lambda:** Las funciones lambda, también conocidas como funciones anónimas, son funciones compactas en una sola línea, que no requieren una declaración formal con `def`. Se definen utilizando la palabra clave `lambda` y son útiles para operaciones simples que comunmente pueden ser expresadas en una sola línea de código. Las funciones lambda son como ayudantes prácticos y ágiles que pueden servirte rápidamente sin mucha ceremonia.
2. **Clases:** Imagina un mundo en el que puedes crear tus propios superhéroes con poderes y habilidades únicas. Eso es exactamente lo que te permiten hacer las clases en la programación. Una clase es como un plano que define las propiedades (atributos) y métodos (superpoderes) de tus superhéroes personalizados (objetos). Encapsula datos y comportamientos relacionados en una sola entidad. Con las clases, puedes crear múltiples instancias (copias individuales de superhéroes) que comparten las mismas atributos y habilidades definidas por la clase. Aquí tienes algunos detalles técnicos adicionales sobre las clases en Python:
- ▶ **Plantillas Poderosas:** En la POO, las clases actúan como plantillas para crear objetos. Definen los atributos (variables) que describen el estado del objeto y los métodos (funciones) que definen su comportamiento. Es como diseñar el plano de un automóvil antes de ensamblar varios de ellos. Las clases se definen utilizando la instrucción `class`.
 - ▶ **Constructores:** Los constructores son métodos especiales utilizados para inicializar objetos. Se llaman cuando se crea un objeto y te permiten configurar el estado o los valores iniciales de los atributos del objeto. Los constructores aseguran que los objetos se inicialicen correctamente y estén listos para ser utilizados. Es como alistar un molde para crear objetos con configuraciones predefinidas.
 - ▶ **Atributos:** En la POO, los objetos tienen atributos que almacenan datos. Los atributos representan las características o propiedades de un objeto. Pueden ser

variables asociadas con un objeto y contienen información. Piensa en ellas como las diferentes características que definen la identidad de un objeto, como color, tamaño o tipo.

- **Métodos:** Los métodos son funciones definidas dentro de una clase y se utilizan para realizar operaciones o acciones relacionadas con el estado del objeto. Pueden manipular los datos del objeto, interactuar con otros objetos o proporcionar funcionalidades específicas dependiendo del contexto en el que se desenvuelve el objeto. Es como si un objeto tuviera habilidades o comportamientos que puede realizar, brindándole vida activa.

Comenzar con funciones y clases te dará experiencia práctica en escribir código reutilizable y modular. Te permitirá descomponer tareas complejas en partes más pequeñas y manejables, y proporcionará una base sólida para una mayor exploración en el emocionante mundo de la programación. ¡Así que ponte el delantal y la capa, y sumérgete en las delicias de las funciones gourmet y las aventuras de las clases súper-heroicas!

6.2 Funciones

¡Vale! ¡Comencemos con algunos ejemplos de código para ver las funciones en acción! Considera usar ideas relacionadas con redes sociales para ilustrar los elementos que enumeraste anteriormente. Podrías usar datos curiosos de las historias de Harry Potter, así como funcionalidades aleatorias disponibles en Python.⁹⁶

¡Claro! A continuación, se presentan los ejemplos.

- **Solucionadores Especializados:** Aquí tienes algunos ejemplos para ilustrar este concepto.

```

1 # Ejemplo 1: Etiquetando Amigos
2 import random
3
4 # Define una función para etiquetar aleatoriamente a un amigo mediante 'def'
5 def etiquetar_amigo():
6     amigos = ["Harry", "Hermione", "Draco", "Ron", "Luna"]
7     amigo_etiquetado = random.choice(amigos)
8     # Muestra el resultado
9     print("Has etiquetado a un amigo:", amigo_etiquetado)
10
11 # Uso
12 print("\n=== Etiquetando a un amigo ===")
13 for _ in range(5):
14     etiquetar_amigo()

```

En este ejemplo, la función `etiquetar_amigo()` se define utilizando la instrucción `def` sin parámetros de entrada, indicados por los paréntesis vacíos `()`. Esta función sirve como un solucionador especializado para el problema de seleccionar y etiquetar aleatoriamente a un amigo de la red del usuario. Utiliza la función `random.choice()` de la librería `random` para realizar esta tarea.

Al llamar a la función `etiquetar_amigo()` varias veces dentro de un bucle, puedes etiquetar repetidamente a diferentes amigos, reutilizando efectivamente el código y ahorrando líneas de código. Este enfoque encapsula la lógica de selección de amigos y etiquetado en una función reutilizable, promoviendo la modularidad y el mantenimiento del código. Al encapsular esta funcionalidad dentro de una función, puedes invocarla fácilmente cuando sea necesario, evitando la necesidad de duplicar el código en todo el programa. Así mismo, puedes actualizar o modificar su código fácilmente cada vez que sea necesario.

```
Has etiquetado a un amigo: Hermione
Has etiquetado a un amigo: Luna
Has etiquetado a un amigo: Harry
Has etiquetado a un amigo: Ron
Has etiquetado a un amigo: Luna
```

```
1 # Ejemplo 2: Recibiendo Notificaciones
2 import datetime, time
3 # Define una función para obtener la fecha y hora de la notificación
4 def obtener_fecha_hora_notificacion():
5     fecha_hora_actual = datetime.datetime.now()
6     fecha_hora_notificacion = fecha_hora_actual.strftime("%d/%b/%Y %H:%M:%S")
7     # Muestra el resultado
8     print("Nueva notificación recibida a las:", fecha_hora_notificacion)
9     time.sleep(random.randint(0, 10)) # Espera unos segundos
10
11 # Uso
12 print("\n=== Obtener notificación ===")
13 for _ in range(3):
14     obtener_fecha_hora_notificacion()
```

En este ejemplo, la función `obtener_fecha_hora_notificacion()` obtiene la fecha y hora actual y la formatea como `'dd/mm/yyyy hh:mm:ss'`. Esta función sirve como un solucionador especializado para el problema de extraer la marca de tiempo de una notificación recibida. Para lograr esta tarea, utiliza la función `datetime.datetime.now()` de la librería `datetime` para obtener la fecha y hora actual y el método `strftime()` para formatearla según el formato especificado. La fecha y hora de notificación formateada resultante se muestra mediante la salida en pantalla actualizada.

Al llamar a la función `obtener_fecha_hora_notificacion()` varias veces en un bucle, puedes simular la recepción de notificaciones en diferentes momentos, reutilizando efectivamente el código sin necesidad de repetir las mismas líneas de código. De nuevo, así se promueve la reutilización del código, eliminando redundancias y garantizando un código más eficiente y conciso.

```
=== Obtener notificación ===
Nueva notificación recibida a las: 23/jul/2023 02:59:47
Nueva notificación recibida a las: 23/jul/2023 02:59:55
Nueva notificación recibida a las: 23/jul/2023 03:00:01
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/ye29dv4m>

- **Valores de Retorno:** A continuación, se presentan algunos ejemplos para ilustrar este concepto.

```
1 # Ejemplo 1: Obtener Información del Usuario
2 def obtener_informacion_usuario():
3     # Simula la obtención de información del usuario desde una plataforma de redes
4     # sociales
5     nombre = "Harry Potter"
6     edad = 17
7     return nombre, edad
8
9 # Ejemplo 2: Obtener Foto de Perfil
10 def obtener_foto_perfil():
11     # Simula la obtención de la foto de perfil del usuario
12     url_foto = "https://twitter.com/Official_HPWiki/photo"
13     return url_foto
14
15 # Ejemplo 3: Obtener Cantidad de Amigos
16 def obtener_cantidad_amigos():
17     # Simula la obtención del número de amigos en la red del usuario
18     amigos = ["Hermione", "Ron", "Neville", "Luna"]
19     return len(amigos)
20
21 # Función Principal
22 def main():
23     # Obtener datos del perfil del usuario
24     nombre, edad = obtener_informacion_usuario()
25     url_foto = obtener_foto_perfil()
26     cantidad_amigos = obtener_cantidad_amigos()
```

```

27     # Mostrar el perfil del usuario
28     print("==== Información del Usuario ====")
29     print("Nombre:", nombre)
30     print("Edad:", edad)
31     print("Foto:", url_foto)
32     print("Cantidad de Amigos:", cantidad_amigos)
33
34 # Llamar a la función principal
35 mail()

```

En el Ejemplo 1, la función `obtener_informacion_usuario()` simula la obtención de información del usuario desde una plataforma de redes sociales. En este ejemplo, asumimos que obtenemos el `nombre` y la `edad` del usuario. Al utilizar la declaración `return`, la función puede devolver simultáneamente múltiples valores, en este caso, una cadena y un número.

En el Ejemplo 2, la función `obtener_foto_perfil()` se define para simular la obtención de la foto de perfil del usuario desde una plataforma de redes sociales; aquí, asumimos que devuelve la URL de la foto, es decir, la cadena `url_foto`.

En el Ejemplo 3, la función `obtener_cantidad_amigos()` se define para simular la obtención del número de amigos que tiene un usuario en una plataforma de redes sociales. La instrucción `return` en este caso devuelve la cantidad de amigos, que corresponde a la longitud de la lista `amigos` obtenida mediante la función `len()`.

Por último, la función principal, usualmente llamada `main()` en inglés, determina el inicio del programa. Es una práctica común en Python tener una función `main()` que actúe como controlador central para ejecutar la lógica principal del programa.

En el ejemplo de código proporcionado, la función `main()` coordina la ejecución de las otras funciones para obtener la información del usuario, la URL de la foto de perfil y la cantidad de amigos. Asigna los valores retornados a variables (`nombre`, `edad`, `url_foto`, `cantidad_amigos`) e imprime el perfil del usuario en la consola.

Tener una función `main()` facilita la organización y legibilidad del código. Separa la lógica de control principal del programa de las definiciones de funciones, lo que facilita la comprensión y el mantenimiento del código. Además, permite una mejor estructura y modularidad del código, promoviendo buenas prácticas de programación.

```

==== Información del Usuario ====
Nombre: Harry Potter
Edad: 17
Foto: https://twitter.com/Official_HPWiki/photo
Cantidad de Amigos: 4

```

Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/2p8bj2k9>

► **Parámetros:** Aquí hay algunos ejemplos para ilustrar este concepto.

```

1 import random
2
3 # Listas predefinidas de usuarios, publicaciones, IDs y comentarios
4 usuarios = ["Harry Potter", "Hermione Granger", "Ron Weasley", "Luna Lovegood"]
5 publicaciones = {
6     123: "Juro solemnemente que mis intenciones no son buenas.",
7     456: "Se puede encontrar felicidad incluso en los momentos más oscuros, si uno
8         recuerda encender la luz.",
9     789: "Son nuestras elecciones, Harry, las que muestran lo que realmente somos,
10        mucho más que nuestras habilidades.",
11    101112: "Todos tenemos luz y oscuridad en nuestro interior. Lo que importa es
12        en cual parte decidimos actuar.",
13 }
14 comentarios = [
15     ("Hermione Granger", "¡Esa es una frase brillante!"),
16     ("Ron Weasley", "¡No podría estar más de acuerdo, colega!"),
17     ("Luna Lovegood", "Tanta sabiduría en esas palabras."),
18     ("Harry Potter", "Me alegra que estas frases resuenen con todos."),
19 ]
20
21 # Función para enviar una solicitud de amistad
22 def enviar_solicitud_amistad(remitente, receptor):
23     print(f"[{remitente}] ha enviado una solicitud de amistad a [{receptor}]")
24
25 # Función para imprimir una publicación
26 def imprimir_publicacion(id_publicacion):
27     etiqueta = ' ID de Publicación: %s ' % (id_publicacion)
28     print(f"{etiqueta: {'-'}{'^'}{80}}")
29     print(f"{publicaciones[id_publicacion]}")
30     print(f'-'*80)
31
32 # Función para dar "me gusta" a una publicación
33 def dar_me_gusta(usuario, id_publicacion):
34     print(f"{usuario} dio 'me gusta' a la publicación con ID {id_publicacion}")
35
36 # Función para comentar en una publicación
37 def comentar_en_publicacion(usuario, id_publicacion, comentario):
38     print(f"{usuario} comentó en la publicación con ID {id_publicacion}: {
39         comentario}")

```

```

37 # Función principal
38 def main():
39     # Repetir interacciones varias veces
40     for _ in range(5):
41         remitente = random.choice(usuarios)
42         receptor = random.choice(usuarios)
43         id_publicacion = random.choice([id for id in publicaciones.keys()])
44         comentador, comentario = random.choice(comentarios)
45
46         imprimir_publicacion(id_publicacion)
47         dar_me_gusta(remitente, id_publicacion)
48         comentar_en_publicacion(comentador, id_publicacion, comentario)
49         enviar_solicitud_amistad(remitente, receptor)
50         print()
51
52 # Llamar a la función principal
53 main()

```

En el primer ejemplo, la función `enviar_solicitud_amistad()` toma dos parámetros: `remitente` y `receptor`. Estos parámetros representan a los personajes de Harry Potter involucrados en la solicitud de amistad. Al pasar los valores adecuados a la función, puedes simular el envío de una solicitud de amistad de un personaje a otro.

El segundo ejemplo, `dar_me_gusta()`, demuestra una función que toma los parámetros `usuario` e `id_publicacion`. El parámetro `usuario` representa al personaje que da “me gusta” a la publicación, y el parámetro `id_publicacion` especifica el ID de la publicación objetivo. Al proporcionar los valores necesarios, puedes simular que un personaje da “me gusta” a una publicación específica.

La función `comentar_en_publicacion()` acepta los parámetros `usuario`, `id_publicacion` y `comentario`. Estos parámetros permiten especificar el usuario que está comentando, el ID de la publicación en la que se hace el comentario y el contenido del comentario. Al pasar los valores adecuados a la función, puedes simular que un personaje comenta en una publicación en particular con un comentario específico.

Por último, la función `main()` utiliza un bucle `for` para repetir las acciones varias veces. En cada iteración, la función selecciona aleatoriamente un remitente y un receptor de la lista de usuarios, una publicación de la lista de publicaciones, un ID de la lista de IDs y un comentario de la lista de comentarios. Luego, llama a las funciones apropiadas (`enviar_solicitud_amistad()`, `dar_me_gusta()` y `comentar_en_publicacion()`) con los valores elegidos al azar.

Al repetir este proceso, el código simula el envío de solicitudes de amistad, dar “me gusta” a publicaciones y comentar en publicaciones con interacciones aleatorias.

Esto muestra cómo los parámetros permiten proporcionar diferentes valores a las funciones, lo que las hace adaptables y reutilizables en diversas situaciones. A continuación una posible salida en pantalla:

```
----- ID de la Publicación: 789 -----  
Son nuestras elecciones, Harry, las que muestran lo que realmente somos, mucho  
más que nuestras habilidades.
```

```
-----  
Ron Weasley dio "me gusta" a la publicación con ID 789  
Luna Lovegood comentó en la publicación con ID 789: Tanta sabiduría en esas  
palabras.  
[Ron Weasley] ha enviado una solicitud de amistad a [Ron Weasley]
```

```
----- ID de la Publicación: 123 -----  
Juro solemnemente que mis intenciones no son buenas.  
-----  
Luna Lovegood dio "me gusta" a la publicación con ID 123  
Luna Lovegood comentó en la publicación con ID 123: Tanta sabiduría en esas  
palabras.  
[Luna Lovegood] ha enviado una solicitud de amistad a [Hermione Granger]
```

```
----- ID de la Publicación: 456 -----  
La felicidad se puede encontrar incluso en los momentos más oscuros, si uno  
solo recuerda encender la luz.  
-----  
Hermione Granger dio "me gusta" a la publicación con ID 456  
Luna Lovegood comentó en la publicación con ID 456: Tanta sabiduría en esas  
palabras.  
[Hermione Granger] ha enviado una solicitud de amistad a [Ron Weasley]
```

```
----- ID de la Publicación: 101112 -----  
Todos tenemos luz y oscuridad en nuestro interior. Lo que importa es en cual  
parte decidimos actuar.  
-----  
Luna Lovegood dio "me gusta" a la publicación con ID 101112  
Harry Potter comentó en la publicación con ID 101112: Me alegra que estas  
frases resuenen con todos.  
[Luna Lovegood] ha enviado una solicitud de amistad a [Hermione Granger]
```



```

----- ID de la Publicación: 123 -----
Juro solemnemente que mis intenciones no son buenas.
-----

Luna Lovegood dio "me gusta" a la publicación con ID 123
Hermione Granger comentó en la publicación con ID 123: ¡Esa es una frase
brillante!

[Luna Lovegood] ha enviado una solicitud de amistad a [Harry Potter]

```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/mtu2uha9>

- **Funciones Lambda:** A continuación se presentan algunos ejemplos para ilustrar este concepto.

```

1 # Ejemplo 1: Ordenar amigos por longitud de nombre
2 amigos = ["Harry Potter", "Hermione Granger", "Ron Weasley", "Draco Malfoy"]
3
4 # Ordenar la lista de amigos según la longitud de sus nombres usando una función
  lambda
5 amigos_ordenados = sorted(amigos, key=lambda x: len(x))
6
7 # Imprimir la lista de amigos ordenados
8 print("\n=== Amigos Ordenados por Longitud de Nombre ===")
9 for amigo in amigos_ordenados:
10     print(f"{amigo} ({len(amigo)})")
11
12 # Ejemplo 2: Filtrar amigos con nombres que comienzan con 'H'
13 amigos = ["Harry Potter", "Hermione Granger", "Ron Weasley", "Draco Malfoy"]
14
15 # Filtrar la lista de amigos para incluir solo aquellos con nombres que comienzan
  con 'H' usando una función lambda
16 amigos_filtrados = list(filter(lambda x: x.startswith('H'), amigos))
17
18 # Imprimir la lista de amigos filtrados
19 print("\n=== Amigos con Nombres que Comienzan con 'H' ===")
20 for amigo in amigos_filtrados:
21     print(amigo)

```

El Ejemplo 1 demuestra el uso de una función lambda como parámetro **key** en la función `sorted()`. La función lambda `lambda x: len(x)` toma el nombre de un amigo como entrada `x` y devuelve la longitud del nombre utilizando `len(x)`. Esta función lambda sirve como base para ordenar la lista de amigos según la longitud de sus nombres.

El Ejemplo 2 muestra el uso de una función lambda como condición de filtrado en la función `filter()`. La función lambda `lambda x: x.startswith('H')` toma el nombre de un amigo como entrada `x` y devuelve un valor booleano que indica si el nombre comienza con 'H', utilizando `startswith('H')`. Esta función lambda se utiliza para filtrar la lista de amigos e incluir solo aquellos cuyos nombres comienzan con 'H'.

```
=== Amigos Ordenados por Longitud de Nombre ===
Ron Weasley (11)
Harry Potter (12)
Draco Malfoy (12)
Hermione Granger (16)

=== Amigos con Nombres que Comienzan con 'H' ===
Harry Potter
Hermione Granger
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/3jkk8meu>

Las funciones lambda son funciones compactas que son útiles para operaciones simples que puede escribirse en una sola línea. Proporcionan una forma conveniente de definir funciones sobre la marcha, sin necesidad de una declaración formal con la instrucción `def`.

6.3 Clases y objetos

¡Muy bien! ¡Continuemos con algunos ejemplos de código para ver las clases en acción! Considera usar preguntas de películas y personajes clásicos para ilustrar los elementos que mencionaste anteriormente, incluyendo funcionalidades aleatorias de Python.⁹⁷

¡Claro que sí! A continuación, se presentan algunos ejemplos de código que demuestran el uso de clases en el contexto de preguntas sobre películas.

- **Plantillas Poderosas:** Las clases son como plantillas poderosas para crear objetos con estructuras predefinidas. Definen la estructura y el comportamiento general de los objetos. Una vez que has definido una clase, puedes usarla para crear tantas instancias u objetos de esa clase como necesites. Cada objeto tendrá su propio conjunto de datos y se comportará de acuerdo con la plantilla definida por la clase.

Para crear una clase en Python, utiliza la palabra clave `class` seguida del nombre de la clase y dos puntos. Para definir métodos dentro de la clase, utiliza la palabra clave `def` seguida del nombre del método, paréntesis y cualquier parámetro necesario. El primer parámetro de cada método debe ser `self`, que almacena una referencia a la instancia particular que ejecutará el método. ¿Observas la sangría o tabulado hacia la derecha? Esto indica que los métodos forman parte de la definición de la clase.

Así que sin más preámbulos, ¡empecemos a crear una poderosa plantilla para una franquicia de películas!

```

1 class FranquiciaPelículas:
2     # Método setter
3     def set_franquicia(self, nombre, peliculas):
4         self.nombre = nombre
5         if peliculas > 0:
6             self.peliculas = peliculas
7         else:
8             print(f"Advertencia: El número de películas de la franquicia '{nombre}'
9             debe ser mayor que 0.")
10            self.peliculas = 1
11
12    # Método getter
13    def get_franquicia(self):
14        return self.nombre, self.peliculas
15
16 def main():
17     # Crear franquicias de películas
18     harry_potter = FranquiciaPelículas()
19     harry_potter.set_franquicia("Harry Potter", 8)
20
21     star_wars = FranquiciaPelículas()
22     star_wars.set_franquicia("Star Wars", 9)
23
24     avatar = FranquiciaPelículas()
25     avatar.set_franquicia("Avatar", -2)
26
27     # Mostrar franquicias de películas
28     print("\n----- Franquicias de Películas -----")
29     print(f"{'Nombre':^14}\t|{'Películas':^10}")
30     print("-----")
31     franquicias = [harry_potter, star_wars, avatar]
32     for franquicia in franquicias:
33         nombre, peliculas = franquicia.get_franquicia()
34         print(f"{'nombre:14'}\t|{'peliculas:10'}")
35
36 # Llamar a la función principal
37 main()

```

En este ejemplo, definimos la clase `FranquiciaPelículas` con métodos para establecer y obtener el nombre y el número de películas de la franquicia. Los métodos *setters* y *getters* se definen con ese propósito. Proporcionan acceso controlado a los atributos de un objeto al encapsularlos dentro de la clase.

Los *setters* (o métodos modificadores) se utilizan para establecer el valor de un atributo. Permiten validar la entrada o realizar operaciones adicionales antes de asignar un valor al atributo. En el código, `set_franquicia()` es un método modificador que establece los valores de los atributos `nombre` y `películas` de la clase `FranquiciaPelículas`, respectivamente. Observa que el método además valida que la cantidad de películas sea un valor positivo mayor que cero.

Los *getters* (o métodos examinadores) se utilizan para obtener el valor de un atributo. Proporcionan acceso al valor del atributo desde fuera de la clase. En el código, `get_franquicia()` es un método examinador que devuelve los valores de los atributos `nombre` y `películas` de la clase `FranquiciaPelículas`, respectivamente.

La función `main()` crea tres objetos de `FranquiciaPelículas` y asigna los datos para cada uno. En caso de que una franquicia tenga un número negativo de películas, se corrige automáticamente para tener una. Luego, muestra las franquicias en una tabla en la consola con separadores decorativos. La tabla se formatea utilizando espacios para alinear correctamente las columnas.

Observa que en el bucle `for` del código anterior, la llamada al método `franquicia.get_franquicia()` recupera y muestra los datos correspondientes de cada franquicia. Es en este sentido que cada objeto se comporta según la plantilla de su clase, pero cada uno de manera única.

```
Advertencia: El número de películas de la franquicia 'Avatar' debe ser mayor
que 0.
```

```
----- Franquicias de Películas -----
```

```
Nombre      | Películas
```

```
-----
```

```
Harry Potter |      8
```

```
Star Wars    |      9
```

```
Avatar       |      1
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/4ab5dd89>

- **Constructores:** Los constructores son métodos especiales que se llaman automáticamente cuando se crea un objeto. Inicializan los atributos del objeto o realizan cualquier configuración necesaria. Crearemos una clase llamada `Personaje` con un constructor para inicializar su nombre y la película en la que figuraron.

```

1 import random
2
3 class Personaje:
4     # El método constructor
5     def __init__(self, nombre, pelicula):
6         self.nombre = nombre
7         self.pelicula = pelicula
8
9     # Un método para presentar el Personaje al usuario
10    def presentarse(self):
11        saludos = ["¡Hola!", "¿Qué tal?", "¡Buenas!", "¡Saludos!", "¡Un placer!"]
12        saludo = random.choice(saludos)
13        print(f"{saludo} Soy {self.nombre} de la película {self.pelicula}.")
14
15    def main():
16        # Crea varias instancias de Personaje
17        dorothea = Personaje("Dorothea", "El Mago de Oz")
18        james = Personaje("James Bond", "Serie 007")
19        sarah = Personaje("Sarah Connor", "Terminator")
20        indy = Personaje("Indiana Jones", "Indiana Jones")
21        leia = Personaje("Princesa Leia", "Star Wars")
22
23        # Mostrar cómo todas las instancias pueden presentarse
24        personajes = [dorothea, james, sarah, indy, leia]
25        for personaje in personajes:
26            personaje.presentarse()
27
28    # Llamar a la función principal
29    main()

```

El método `__init__` se utiliza para definir el constructor, el cual toma `self`, `nombre` y `pelicula` como parámetros. Dentro del constructor, los atributos `nombre` y `pelicula` se les asignan los valores pasados como parámetros, utilizando la palabra clave reservada `self`, la cual hace referencia al objeto que se está creando. Esto asegura que cada instancia de la clase `Personaje` tenga su propio `nombre` y `pelicula` independientes. Es una práctica común utilizar los mismos nombres para los parámetros y los atributos, aunque también se pueden nombrar de manera diferente.

Además, la clase implementa el método `presentarse()` que selecciona aleatoriamente un saludo diferente de la lista `saludos` y lo utiliza para presentar al personaje. Este comportamiento dinámico aporta un elemento de sorpresa a las interacciones del personaje con el usuario.

Asimismo, la creación de objetos de `Personaje` se realiza dentro de la función `main()`. Se definen las variables de objeto y los atributos se pasan como parámetros al llamar al constructor. Posteriormente, se crea una lista de personajes y se utiliza un bucle para realizar sus presentaciones:

```
iHola! Soy Dorothy de la película El Mago de Oz.
¿Qué tal? Soy James Bond de la película Serie 007.
¡Buenas! Soy Sarah Connor de la película Terminator.
¡Un placer! Soy Indiana Jones de la película Indiana Jones.
¡Buenas! Soy Princesa Leia de la película Star Wars.
```

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/mt8fbrzw>

- **Atributos:** Los atributos son características o propiedades de un objeto. Contienen los datos que determinan el estado de un objeto en un momento específico durante la ejecución de un programa. En términos computacionales, los atributos se pueden pensar como variables asociadas a un objeto. Por ejemplo, definamos atributos para una clase `Actor`:

```
1 class Actor:
2     def __init__(self, nombre, pais, oscar):
3         # Atributos no mutables
4         self.nombre = nombre
5         self.pais = pais
6         # Atributo mutable
7         self.oscar = oscar
8
9     def ganar_oscar(self):
10        if not self.oscar:
11            self.oscar = True
12            print(f"¡{self.nombre} acaba de ganar un Oscar!")
13        else:
14            print(f"{self.nombre} ya tiene un Oscar.")
15
16    def __str__(self):
17        return f"{self.nombre:20}\t{self.pais:10}\t{'Sí' if self.oscar else 'No'}"
18
```

En este código, los atributos `nombre` y `pais` se consideran no mutables porque no tiene sentido que el nombre o el país de un actor cambien una vez que se han establecido por el constructor. Estos atributos se establecen una vez durante la creación del objeto y permanecen constantes a lo largo de la vida del objeto.

Por otro lado, el atributo `oscar` es mutable porque representa el estado del Oscar del actor, que puede cambiar de `False` a `True` cuando el actor gana un Oscar. El método `ganar_oscar()` permite modificar este atributo, pero no se puede cambiar de nuevo a `False`, ya que no es posible que a un actor se le quite el premio una vez que se le ha otorgado.

Por último, observe que esta clase sobrescribe el método predefinido `__str()` para obtener una representación en forma de cadena del objeto `Actor`, que sea adecuada para mostrarlo en la consola. El método `__str()` se llama cuando el objeto necesita convertirse en una cadena, como al imprimirlo o utilizar opciones de formateo. Al definir este método, podemos personalizar el formato de salida del objeto `Actor`, en este caso, para mostrar el nombre, el país y el estado del Oscar en una estructura de tabla formateada.

Ahora podemos escribir la función `main()` para crear una lista de actores, mostrar su información en una tabla, actualizar el estado del Oscar para algunos actores e imprimir la tabla actualizada:

```

1 def main():
2     actores = [
3         Actor("Meryl Streep", "Estados Unidos", True),
4         Actor("Daniel Day-Lewis", "Reino Unido", True),
5         Actor("Cate Blanchett", "Australia", True),
6         Actor("Leonardo DiCaprio", "Estados Unidos", False),
7         Actor("Anthony Hopkins", "Reino Unido", True),
8         Actor("Kate Winslet", "Reino Unido", True),
9         Actor("Denzel Washington", "Estados Unidos", True),
10        Actor("Julianne Moore", "Estados Unidos", True),
11        Actor("Tom Hanks", "Estados Unidos", False),
12        Actor("Natalie Portman", "Israel", False),
13    ]
14
15    print("\n===== Información de los Actores =====")
16    print(f"{'Nombre':20}\t{'País':10}\t{'Oscar?'}")
17    print("="*45)
18    for actor in actores:
19        print(actor)
20
21    print("\n===== Actualizando Estado del Oscar =====")
22    actores[3].ganar_oscar()
23    actores[4].ganar_oscar()
24    actores[8].ganar_oscar()
25

```

```

26 print("\n===== Información Actualizada de los Actores =====")
27 print("Nombre\tPaís\tOscar?")
28 print("-"*45)
29 for actor in actores:
30     print(actor)
31

```

===== Información de los Actores =====

Nombre	País	Oscar?
--------	------	--------

=====

Meryl Streep	Estados Unidos	Sí
Daniel Day-Lewis	Reino Unido	Sí
Cate Blanchett	Australia	Sí
Leonardo DiCaprio	Estados Unidos	No
Anthony Hopkins	Reino Unido	Sí
Kate Winslet	Reino Unido	Sí
Denzel Washington	Estados Unidos	Sí
Julianne Moore	Estados Unidos	Sí
Tom Hanks	Estados Unidos	No
Natalie Portman	Israel	No

===== Actualizando Estado del Oscar =====

¡Leonardo DiCaprio acaba de ganar un Oscar!

Anthony Hopkins ya tiene un Oscar.

¡Tom Hanks acaba de ganar un Oscar!

== Información Actualizada de los Actores ==

Nombre	País	Oscar?
--------	------	--------

=====

Meryl Streep	Estados Unidos	Sí
Daniel Day-Lewis	Reino Unido	Sí
Cate Blanchett	Australia	Sí
Leonardo DiCaprio	Estados Unidos	Sí
Anthony Hopkins	Reino Unido	Sí
Kate Winslet	Reino Unido	Sí
Denzel Washington	Estados Unidos	Sí
Julianne Moore	Estados Unidos	Sí
Tom Hanks	Estados Unidos	Sí
Natalie Portman	Israel	No

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/j4fnxc5m>

- **Métodos:** Los métodos son funciones definidas dentro de una clase que realizan acciones u operaciones específicas en objetos, utilizando sus propios atributos o datos externos proporcionados por otros objetos o funciones a través de parámetros de entrada. También pueden devolver datos como resultado de su ejecución. Desde el punto de vista computacional, los métodos en la POO encapsulan la lógica algorítmica que define el comportamiento de un objeto.

Para ilustrar este concepto, creemos una clase llamada `Pelicula` que incluya métodos para procesar sus atributos y proporcionar información útil al usuario. Estos métodos realizan operaciones en los atributos del objeto `Pelicula` para extraer información relevante. Por ejemplo, podemos determinar su duración, obtener la lista de premios Oscar otorgados a la película e identificar el siglo en el que fue producida.

```

1 class Pelicula:
2     # El constructor de Pelicula requiere nombre, año, duración y una lista
   opcional de Oscars
3     def __init__(self, nombre, año, duracion, oscars=None):
4         self.nombre = nombre
5         self.año = año
6         self.duracion = duracion
7         self.oscars = oscars if oscars else [] # Establecer lista vacía si oscars
   == None
8
9     # Agregar un premio Oscar a la película. Método sin retorno con mensaje de
   confirmación
10    def agregar_oscar(self, premio):
11        self.oscars.append(premio)
12        print(f"Se agregó el premio {premio} a {self.nombre}")
13
14    # Devolver la lista de Oscars como una cadena separada por comas
15    def obtener_lista_oscar(self):
16        return ", ".join(self.oscars)
17
18    # Devolver la categoría de duración: "Normal", "Larga" o "Muy larga"
19    def obtener_duración(self):
20        if self.duracion < 90:
21            return "Normal" # < 90 minutos
22        elif self.duracion < 180:
23            return "Larga" # 90-180 minutos
24        else:
25            return "Muy larga" # > 180 minutos
26

```

```

27     # Calcular el siglo basado en el año y devolverlo como una cadena formateada
28     def obtener_siglo(self):
29         siglo = (self.año // 100) + 1
30         return f"Siglo {siglo}"
31
32     # Devolver representación formateada de la película
33     def __str__(self):
34         return f"{self.nombre:<20}\t{self.obtener_siglo():<10}\t{self.
35             obtener_duración():<15}\t{self.obtener_lista_oscar()}"

```

El constructor `__init__` toma parámetros como `nombre`, `año`, `duracion` (en minutos) y una lista de `oscars`. El parámetro `oscars` tiene un valor predeterminado de `None`, lo que permite omitirlo al crear el objeto. Si no se proporciona ningún valor para `oscars`, se establece como una lista vacía para garantizar que el objeto se inicialice correctamente con valores adecuados, incluso si la lista de Oscars no se especifica durante la creación del objeto.

Por otro lado, el método `agregar_oscar` agrega un premio Oscar a la película mediante la adición del premio a la lista de atributos `oscars`. También imprime un mensaje de confirmación para indicar que se agregó el premio.

Además, la clase proporciona varios métodos *getter* que operan sobre los atributos del objeto película y devuelven nueva información. El método `obtener_lista_oscar` recupera la lista de Oscars como una cadena separada por comas al unir los elementos de la lista `oscars`. Esto permite un fácil acceso a los premios que ha recibido la película.

El método `obtener_duración` determina la categoría de duración de la película en función de su atributo `duracion`. Categoriza las películas como “Normal” si su duración es menor a 90 minutos, “Larga” si está entre 90 y 180 minutos y “Muy larga” si supera los 180 minutos. De esta manera provee una descripción concisa de la duración de la película.

El método `obtener_siglo` calcula el siglo en el que se produjo la película en función de su atributo `año`. Divide el año por 100, suma 1 al resultado y devuelve el siglo como una cadena formateada. Esto permite a los usuarios identificar rápidamente el período de tiempo al que pertenece la película.

Por último, el método `__str__` proporciona una representación formateada en cadena del objeto película. Devuelve una cadena que contiene el nombre de la película, el año, la categoría de duración y la lista de Oscars. Esta representación en cadena permite una fácil impresión o visualización concisa de la información de la película.

En resumen, la clase `Pelicula` proporciona métodos convenientes para interactuar con los atributos de un objeto película y obtener información adicional derivada de

esos atributos. Ahora podemos escribir una función `main()` para usar esta clase, crear varios objetos de `Pelicula` y mostrar información útil sobre ellos.

```

1 def main():
2
3     # Las listas de Premios Oscar son solo ilustrativas, no son las reales
4     peliculas = [
5         Pelicula("Titanic", 1997, 194, ["Mejor Película", "Mejor Actor"]),
6         Pelicula("El Señor de los Anillos", 2003, 201, ["Mejor Película"]),
7         Pelicula("Ben-Hur", 1959, 212, ["Mejor Música", "Mejor Actriz"]),
8         Pelicula("Gravedad", 2013, 89, ["Mejor Actor", "Mejor Música"]),
9         Pelicula("Lo que el Viento se Llevó", 1939, 238, ["Mejor Película", "Mejor
10         Música"]),
11         Pelicula("La Forma del Agua", 2017, 123),
12     ]
13
14     print("Lista de Películas más Premiadas:")
15     print("-"*80)
16     print("Nombre\t\t\tAño\t\tDuración\t\tPremios")
17     print("-"*80)
18     for pelicula in peliculas:
19         print(pelicula)
20
21     print("\nModificando Premios:")
22     peliculas[5].agregar_oscar("Mejor Actor")
23     peliculas[5].agregar_oscar("Mejor Actriz")
24
25     print("\nLista de Películas Actualizada:")
26     print("-"*80)
27     print("Nombre\t\t\tAño\t\tDuración\t\tPremios")
28     print("-"*80)
29     for pelicula in peliculas:
30         print(pelicula)
31
32 # Llamar a la función principal
33 main()

```

La función `main()` crea una lista de objetos de películas, cada uno representando una película diferente con atributos como nombre, año, duración (en minutos) y una lista ilustrativa de premios Oscars ganados. En la lista de películas, la película “The Shape of Water” se inicializa sin una lista inicial de premios Oscars, por lo que utiliza el valor predeterminado en el constructor, que es una lista vacía.

El código luego procede a imprimir una tabla de las películas más premiadas. Itera sobre la lista de películas y llama al método `__str__` de cada objeto `Pelicula` para obtener la representación de cadena con formato de columna. Seguidamente, el código modifica los premios para “La Forma del Agua” llamando al método `agregar_oscar()` en el objeto `Pelicula` correspondiente. Se agregan dos nuevos premios, “Mejor Actor” y “Mejor Actriz”, a su lista de premios Oscars.

Finalmente, se imprime la lista de películas actualizada, mostrando los cambios realizados en los premios para “La Forma del Agua”, así como la información del resto de las películas. A continuación se muestra la salida de este programa:

Lista de películas más premiadas:

Nombre	Período	Duración	Oscars
Titanic	Siglo XX	Larga	Mejor Película, Mejor Actor
El Señor de los Anillos	Siglo XXI	Larga	Mejor Película
Ben-Hur	Siglo XX	Larga	Mejor Música, Mejor Actriz
Gravedad	Siglo XXI	Normal	Mejor Actor, Mejor Música
Lo que el Viento se Llevó	Siglo XX	Larga	Mejor Película, Mejor Música
La Forma del Agua	Siglo XXI	Larga	

Modificando Premios:

Agregado premio Mejor Actor a La Forma del Agua

Agregado premio Mejor Actriz a La Forma del Agua

Lista de películas más premiadas:

Nombre	Período	Duración	Oscars
Titanic	Siglo XX	Larga	Mejor Película, Mejor Actor
El Señor de los Anillos	Siglo XXI	Larga	Mejor Película
Ben-Hur	Siglo XX	Larga	Mejor Música, Mejor Actriz
Gravedad	Siglo XXI	Normal	Mejor Actor, Mejor Música
Lo que el Viento se Llevó	Siglo XX	Larga	Mejor Película, Mejor Música
La Forma del Agua	Siglo XXI	Larga	Mejor Actor, Mejor Actriz

↖ Ejecuta paso a paso este código en PythonTutor: <https://tinyurl.com/3pse4vrt>

Estos ejemplos demuestran el uso de clases como plantillas con atributos, métodos y constructores en el contexto de películas clásicas. Al aprovechar el poder de las clases y la programación orientada a objetos, puedes liberar tu creatividad y embarcarte en un emocionante viaje hacia un mundo lleno de posibilidades.

6.4 ¡Manos a la obra!

Sugiere algunos ejercicios de programación divertidos y atractivos para practicar funciones, valores de retorno, parámetros y expresiones lambda. Cada ejercicio debe ir acompañado de una descripción del problema y ejemplos de la salida esperada. Puedes incorporar elementos de matemáticas o aleatoriedad, que requieran importar librerías necesarias para resolver los problemas.⁹⁸

1. **Simulador de Dados:** Crea una función que simule tirar un dado. La función debe tomar el número de lanzamientos del dado como parámetro y devolver una lista con los resultados obtenidos. Ejemplo:

```
Entrada: 5
Salida: [3, 6, 1, 4, 2]
```

2. **Secuencia de Fibonacci:** Escribe una función que genere la secuencia de Fibonacci hasta un número especificado de términos. La función debe tomar el número de términos como parámetro y devolver una lista con la secuencia. Ejemplo:

```
Entrada: 8
Salida: [0, 1, 1, 2, 3, 5, 8, 13]
```

3. **Verificador de Números Primos:** Implementa una función que verifique si un número dado es primo. La función debe tomar el número como parámetro y devolver `True` si es primo, y `False` en caso contrario. Ejemplo:

```
Entrada: 17
Salida: True
```

4. **Verificador de Palíndromos:** Crea una función que determine si una palabra dada es un palíndromo. La función debe tomar la palabra como parámetro y devolver `True` si es un palíndromo, y `False` en caso contrario. Ejemplo:

```
Entrada: "radar"
Salida: True
```

5. **Calculadora de Área de Círculo:** Escribe una función que calcule el área de un círculo dado su radio. La función debe tomar el radio como parámetro y devolver el área calculada. Ejemplo:

```
Entrada: 5
Salida: 78.54
```

6. **Suma de Cuadrados:** Crea una expresión lambda que calcule la suma de los cuadrados de una lista de números. La expresión lambda debe tomar una lista de números como parámetro y devolver la suma de sus cuadrados. Ejemplo:

```
Entrada: [1, 2, 3, 4, 5]
```

```
Salida: 55
```

7. **Contador de Frecuencia de Palabras:** Escribe una función que cuente la frecuencia de cada palabra en una frase dada, sin importar mayúsculas o minúsculas. La función debe tomar la frase como parámetro y devolver un diccionario donde las claves son las palabras y los valores son las frecuencias. Implementa una función de menú que solicite al usuario ingresar una frase y llame a la función de conteo de frecuencia de palabras. Debe mostrar el diccionario resultante de las frecuencias de palabras. Crea una función `main` que llame a la función de menú en un bucle, permitiendo al usuario ejecutar el contador de frecuencia de palabras varias veces hasta que decida salir.

Ejemplo de una llamada a la función:

```
Entrada: "El gato en el sombrero"
```

```
Salida: {'el': 2, 'gato': 1, 'en': 1, 'sombrero': 1}
```

8. **Inversor de Cadenas:** Implementa una función que reverse una cadena dada. La función debe tomar la cadena como parámetro y devolver la cadena invertida. Crea una función de menú que solicite al usuario ingresar una cadena y llame a la función inversora de cadenas. Debe mostrar la cadena invertida. Escribe una función `main` que llame a la función de menú en un bucle, permitiendo al usuario invertir cadenas varias veces hasta que decida salir.

Ejemplo de una llamada a la función:

```
Entrada: "¡Hola, mundo!"
```

```
Salida: "!odnum ,aloHi"
```

9. **Elementos Únicos:** Crea una función que tome una lista de números como entrada y devuelva una nueva lista que contenga solo los elementos únicos de la lista original, en el mismo orden. Desarrolla una función de menú que solicite al usuario ingresar una lista de números y llame a la función de elementos únicos. Debe mostrar la lista resultante de elementos únicos. Diseña una función `main` que llame a la función de menú en un bucle, permitiendo al usuario encontrar elementos únicos de diferentes listas de números hasta que decida salir.

Ejemplo de una llamada a la función:

```
Entrada: [1, 2, 3, 2, 4, 1, 5]
```

```
Salida: [1, 2, 3, 4, 5]
```

10. **Promedio de Notas:** Escribe una función que calcule el promedio de notas de un estudiante dada una lista de notas. La función debe tomar la lista de notas como parámetro y devolver el promedio como un número decimal. Construye una función de menú que solicite al usuario ingresar una lista de notas y llame a la función de promedio de notas. Debe mostrar el promedio calculado. Desarrolla una función `main` que permita al usuario calcular el promedio de notas varias veces hasta que decida salir.

Ejemplo de una llamada a la función:

```
Entrada: [85, 90, 92, 88, 95]
Salida: 90.0
```

11. **Cifrado César:** Implementa una función que encripte un mensaje dado utilizando la técnica de cifrado César. La función debe tomar el mensaje y la cantidad de desplazamiento como parámetros y devolver el mensaje encriptado. Crea una función de menú que solicite al usuario ingresar un mensaje y un valor de desplazamiento, y llame a la función de cifrado César. Debe mostrar el mensaje encriptado. Escribe una función `main` que llame a la función de menú en un bucle, permitiendo al usuario encriptar mensajes utilizando el cifrado César varias veces hasta que decida salir.

Ejemplo de una llamada a la función:

```
Entrada: ("¡Hola, mundo!", 3)
Salida: "¡Krod, pxqgr!"
```

12. **Resolutor de Ecuación Lineal:** Crea una función que resuelva una ecuación lineal de la forma $ax + b = 0$. La función debe tomar los valores de a y b como parámetros y devolver la solución x . Implementa una función de menú que solicite al usuario ingresar los valores de a y b y llame a la función resolutora de ecuación lineal. Debe mostrar la solución x . Escribe una función `main` que llame a la función de menú en un bucle, permitiendo al usuario resolver ecuaciones lineales varias veces hasta que decida salir.

Ejemplo de una llamada a la función:

```
Entrada: (2, 4)
Salida: -2.0
```

¡Vale! Ahora sugiere algunos ejercicios de programación divertidos y atractivos para practicar POO, incluyendo clases, constructores, atributos y métodos. Cada ejercicio debe ir acompañado de una descripción del problema y ejemplos de la salida esperada. Puedes incorporar elementos de matemáticas o aleatoriedad, que requieran importar librerías necesarias para resolver los problemas.⁹⁹

1. **Calculadora de Notas de Estudiantes:** Crea una clase llamada `Estudiante` que encapsule la información de un estudiante, incluyendo su nombre y una lista de notas. Implementa métodos para calcular el promedio de las notas, determinar si el estudiante aprobó (promedio de notas ≥ 60) y mostrar la información del estudiante en pantalla. Además, escribe una función `main` que permita al usuario interactuar con los objetos `Estudiante` a través de un menú. Ejemplo:

► Entrada:

```

1 estudiante1 = Estudiante("Juan Pérez", [75, 80, 90, 65, 70])
2 estudiante1.mostrar_info() # Mostrar información del estudiante
3 print("Promedio de Notas:", estudiante1.calcular_promedio_notas()) # Mostrar
  promedio de notas
4 print("Aprobado:", estudiante1.ha_aprobado()) # Verificar si el estudiante
  aprobó
5

```

► Salida:

```

Nombre del Estudiante: Juan Pérez
Notas: [75, 80, 90, 65, 70]
Promedio de Notas: 76.0
Aprobado: True

```

► Menú:

```

1. Agregar Estudiante
2. Mostrar Información del Estudiante
3. Calcular Promedio de Notas
4. Verificar Estado de Aprobación
5. Salir

Ingresa tu opción:

```

2. **Registro de Transacciones Bancarias:** Crea una clase llamada `CuentaBancaria` que represente una cuenta bancaria con atributos como número de cuenta, nombre del titular de la cuenta y saldo. Implementa métodos para depositar y retirar dinero de la cuenta, así como un método para mostrar el historial de transacciones de la cuenta. El historial de transacciones debe llevar un registro de todos los depósitos y retiros realizados en la cuenta. Escribe una función `main` que permita al usuario interactuar con el objeto `CuentaBancaria` a través de un menú. Ejemplo:

► Entrada:

```

1 cuenta1 = CuentaBancaria("123456789", "Juan Pérez", 1000.0)
2 cuenta1.depositar(500.0) # Depositar $500
3 cuenta1.retirar(200.0) # Retirar $200
4 cuenta1.depositar(300.0) # Depositar $300
5 cuenta1.mostrar_historial_transacciones() # Mostrar historial

```

► Salida:

```

Cuenta: 123456789
Nombre: Juan Pérez
Saldo: $1,600.00
-----
Historial de Transacciones
-----
Depósito:    $500.0
Retiro:      $200.0
Depósito:    $300.0

```

► Menú:

```

1. Depositar
2. Retirar
3. Mostrar Historial de Transacciones
4. Salir

```

Ingresa tu opción:

3. Sistema de Gestión de Inventario:

Crea una clase llamada **Producto** que represente un producto en un sistema de gestión de inventario. La clase debe tener atributos como el ID del producto, nombre, cantidad y precio. Implementa métodos para aumentar o disminuir la cantidad de un producto, calcular el valor total del producto (cantidad × precio) y mostrar la información del producto. Además, escribe una función **main** que permita al usuario interactuar con los objetos **Producto** a través de un menú. Ejemplo:

► Entrada:

```

1 producto1 = Producto("P001", "iPhone", 10, 1000.0)
2 producto1.aumentar_cantidad(5) # Aumentar cantidad en 5
3 producto1.disminuir_cantidad(3) # Disminuir cantidad en 3
4 producto1.mostrar_info() # Mostrar información del producto
5
6 print("Valor Total:", producto1.calcular_valor_total()) # Imprimir total

```

► Salida:

```
ID del Producto: P001
Nombre: iPhone
Cantidad: 12
Precio: $1000.0
Valor Total: $12000.0
```

► Menú:

```
1. Agregar Producto
2. Aumentar Cantidad
3. Disminuir Cantidad
4. Calcular Valor Total
5. Mostrar Información del Producto
6. Salir

Ingresa tu opción:
```

4. **Juego de Trivia de Películas:** Se te ha encargado crear un juego de preguntas y respuestas sobre Películas. El juego deberá poner a prueba el conocimiento del jugador sobre películas populares mediante una serie de preguntas de opción múltiple. Debes codificar las clases necesarias e implementar un sistema de menú para permitir al usuario jugar la Trivia. Aquí están los requisitos:

- a) Crea una clase llamada **Pregunta** que represente una sola pregunta de trivia. Cada pregunta debe tener los siguientes atributos:
 - **texto**: La pregunta que se va a hacer.
 - **opciones**: Una lista de opciones de múltiple elección.
 - **correcta**: El índice de la respuesta correcta en la lista de opciones.
- b) Crea una clase llamada **JuegoTrivia** que gestione el flujo de ejecución del juego. Debe tener los siguientes métodos:
 - **__init__()**: Inicializa el juego, la puntuación y configura las preguntas (crear un conjunto de al menos 10 preguntas).
 - **mostrar_menu()**: Muestra las opciones del menú del juego para el jugador.
 - **comenzar_juego()**: Inicia el juego de trivia.
 - **hacer_pregunta(pregunta)**: Hace una sola pregunta y verifica la respuesta del jugador. Las preguntas deben ser elegidas al azar y no deben repetirse.
 - **mostrar_puntuacion()**: Muestra la puntuación actual del jugador.
 - **finalizar_juego()**: Finaliza el juego y muestra la puntuación final.
- c) Implementa un sistema de menú utilizando la función **main()** para permitir que el jugador interactúe con el juego. El menú debe tener las siguientes opciones:

- ▶ Comenzar juego: Inicia el juego de trivia.
- ▶ Mostrar última puntuación: Muestra la puntuación del juego más reciente jugado durante la ejecución, o "¡Juega primero!" si no se ha jugado ninguno.
- ▶ Salir: Sale del juego.

Ejemplo:

```
=====
Bienvenido a la Trivia de Películas
=====

1. Comenzar juego
2. Mostrar última puntuación
3. Salir
Ingresa tu opción: 1
=====

Pregunta 1:
¿Qué película ganó el Premio de la Academia a la Mejor Película en 2020?

A. Parasite
B. Joker
C. 1917
D. The Irishman

Ingresa tu respuesta: A

¡Correcto!

Pregunta 2:
¿Quién interpretó al personaje de Iron Man en el Universo Cinematográfico de
Marvel?

A. Robert Downey Jr.
B. Chris Evans
C. Chris Hemsworth
D. Mark Ruffalo

Ingresa tu respuesta: B

¡Incorrecto!
```

Pregunta 3:

En la película "The Shawshank Redemption," ¿cuál es el nombre del personaje principal?

- A. Andy Dufresne
- B. Red Redding
- C. Ellis Boyd "Red" Redding
- D. Warden Samuel Norton

Ingresa tu respuesta: C

¡Correcto!

Tu puntuación: 2 de 3

¡Gracias por jugar a la Trivia de Películas!

=====

Bienvenido a la Trivia de Películas

=====

1. Comenzar juego
2. Mostrar última puntuación
3. Salir

Ingresa tu opción:

=====

7.1 ¿Qué sigue a continuación?

¡Ha sido un emocionante viaje hasta hasta el momento! Pero ahora que he dominado los fundamentos de la programación en Python (tipos de datos, variables, entrada/salida de datos por consola, operadores, condicionales, bucles, listas, tuplas, cadenas, diccionarios, funciones, clases y objetos), ¿hacia adonde debería continuar? ¿Podrías guiarme en la dirección correcta? Sugiere algunos planes de estudio para profundizar en diferentes rutas de aprendizaje que me ayuden a convertirme en un experto de la programación en Python. (usa un formato de mapa mental como PlantUML)¹⁰⁰.

¡Muy bien, joven prodigio de Python! Ahora que has conquistado los caminos esenciales de la programación en Python, es hora de elevar tu nivel de dominio de este simpático lenguaje. Prepárate para el emocionante viaje que viene, mientras te propongo planes de estudios fascinantes para continuar con tu aventura en el asombroso mundo de Python. Podrás surcar los reinos del desarrollo web, descubrir los secretos de la ciencia de datos, liberar tus poderes en la automatización y el scripting, sumergirte en el mundo del desarrollo de juegos, abrazar la magia de DevOps, desentrañar los misterios de conceptos avanzados de Python, dominar el arte de las pruebas y depuración, y acceder a los tesoros de los recursos adicionales. Con estas lecciones bajo la manga, ¡serás un virtuoso de la codificación en Python! ¡Adelante, futura estrella de Python!

1. Desarrollo web con Python:

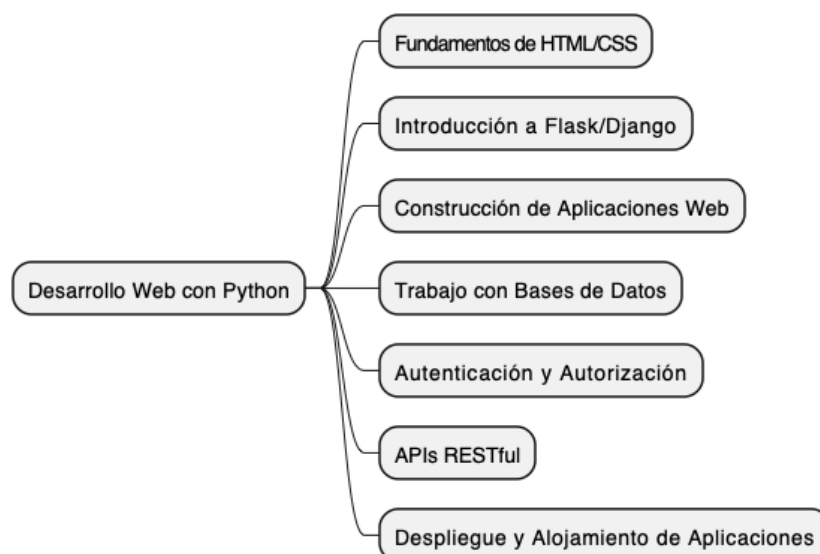


Figura 7.1: Ruta de aprendizaje sugerida para el desarrollo web con Python.

2. Ciencia de datos con Python:

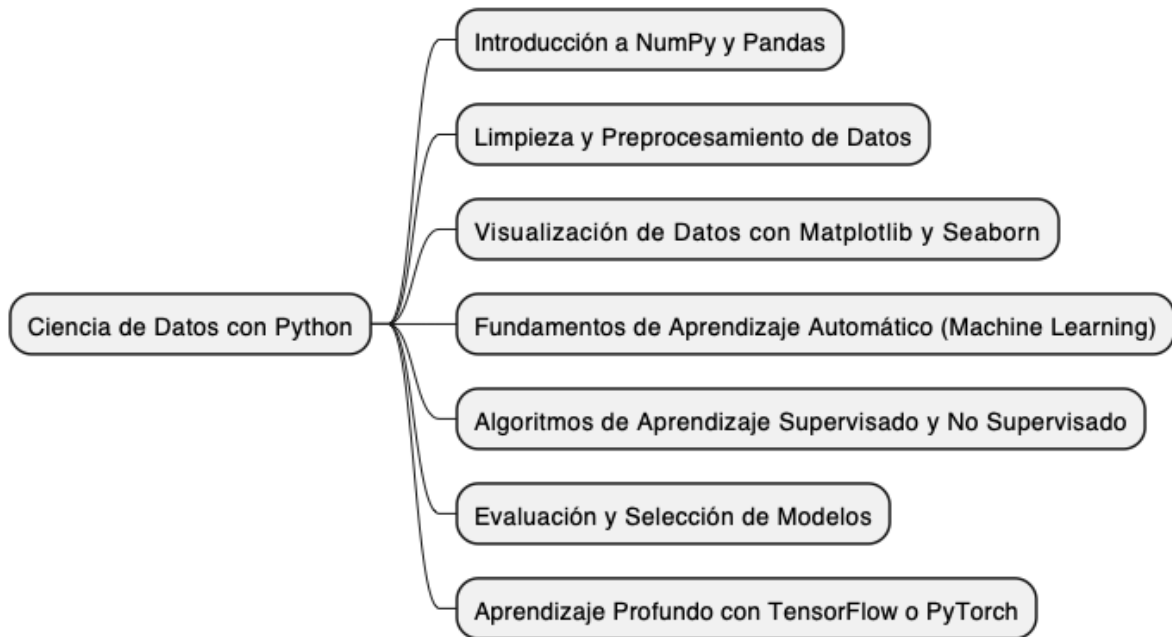


Figura 7.2: Ruta de aprendizaje sugerida para la ciencia de datos con Python.

3. Automatización y scripting con Python:

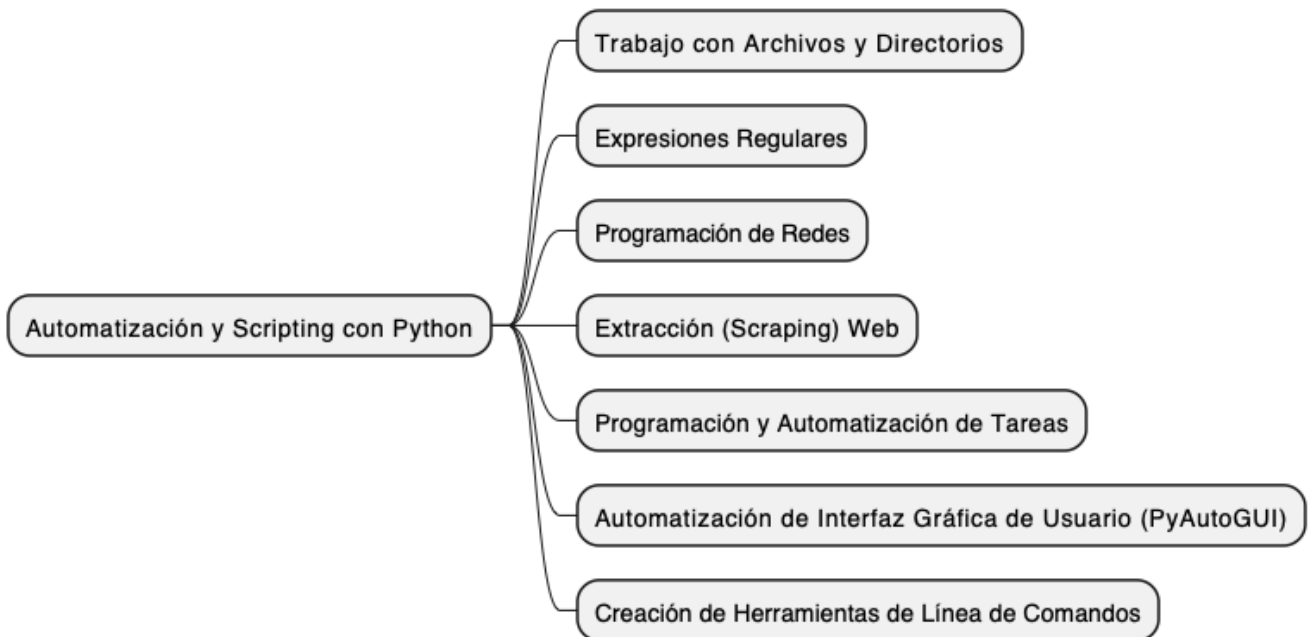


Figura 7.3: Ruta de aprendizaje sugerida para la automatización y el scripting con Python.

4. Desarrollo de juegos con Python:



Figura 7.4: Ruta de aprendizaje sugerida para el desarrollo de juegos con Python.

5. DevOps con Python:



Figura 7.5: Ruta de aprendizaje sugerida para DevOps con Python.

6. Conceptos avanzados de Python:

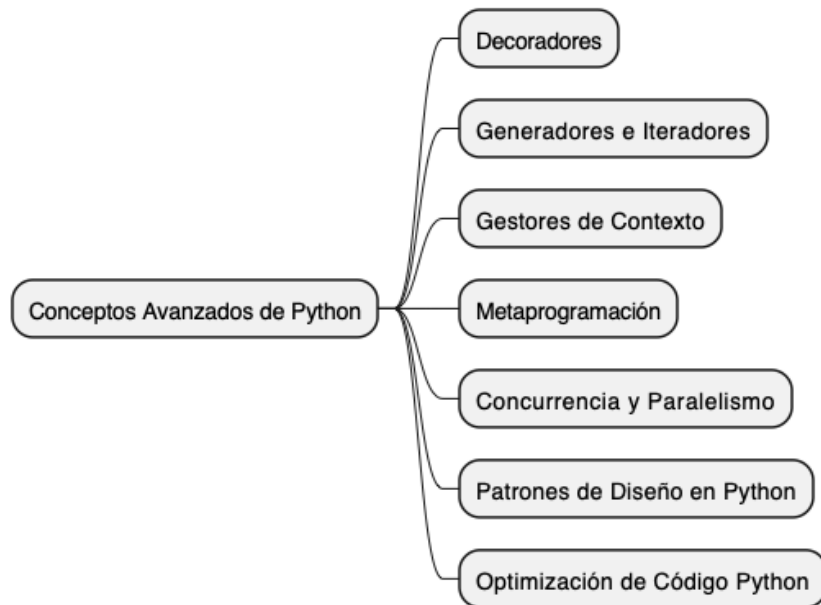


Figura 7.6: Ruta de aprendizaje sugerida para conceptos avanzados de Python.

7. Pruebas y depuración en Python:

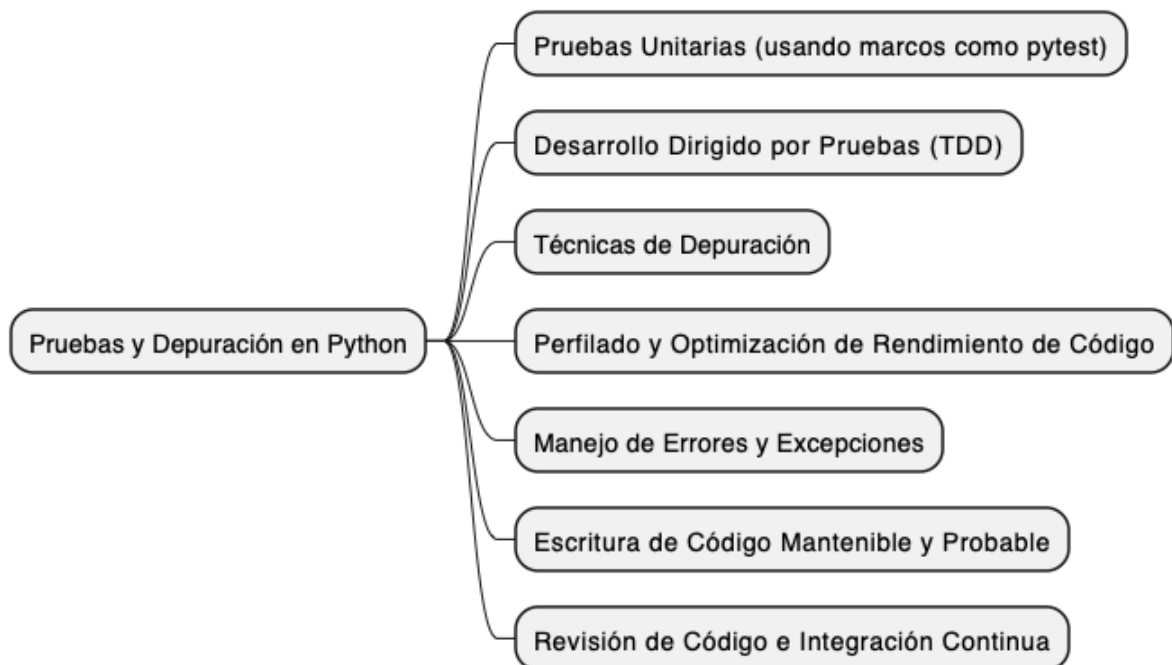


Figura 7.7: Ruta de aprendizaje sugerida para pruebas y depuración en Python.

8. Recursos adicionales:

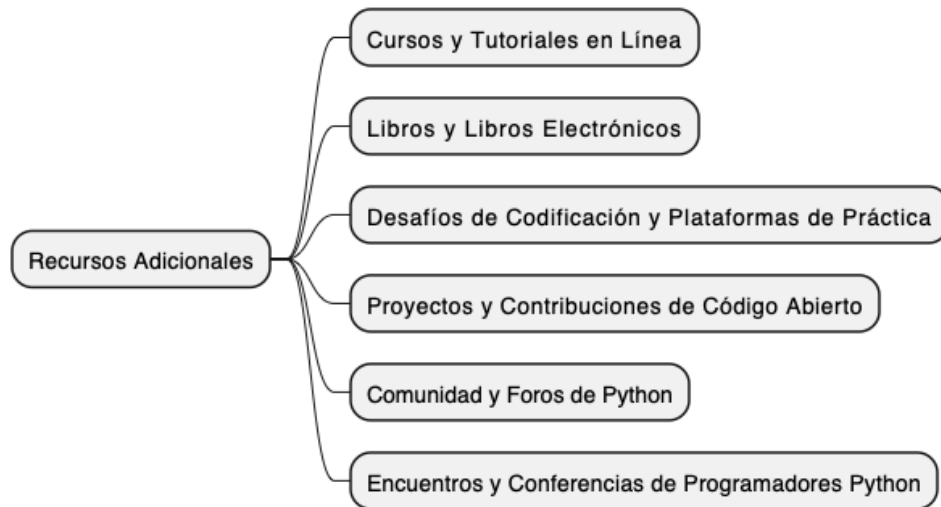


Figura 7.8: Ruta de aprendizaje sugerida para recursos adicionales sobre Python.

Nota: A continuación se muestra la salida generada por ChatGPT para el prompt anterior. Posteriormente, esta salida se importó a draw.io como entrada de PlantUML para generar los diagramas en forma de mapa mental.

```

@startmindmap
* Rutas de Aprendizaje de Python
** Desarrollo Web con Python
*** Fundamentos de HTML/CSS
*** Introducción a Flask/Django
*** Construcción de Aplicaciones Web
*** Trabajo con Bases de Datos
*** Autenticación y Autorización
*** APIs RESTful
*** Despliegue y Alojamiento de Aplicaciones

** Ciencia de Datos con Python
*** Introducción a NumPy y Pandas
*** Limpieza y Preprocesamiento de Datos
*** Visualización de Datos con Matplotlib y Seaborn
*** Fundamentos de Aprendizaje Automático (Machine Learning)
*** Algoritmos de Aprendizaje Supervisado y No Supervisado
*** Evaluación y Selección de Modelos
*** Aprendizaje Profundo con TensorFlow o PyTorch

** Automatización y Scripting con Python
*** Trabajo con Archivos y Directorios
*** Expresiones Regulares
*** Programación de Redes
*** Extracción (Scraping) Web
*** Programación y Automatización de Tareas
*** Automatización de Interfaz Gráfica de Usuario (PyAutoGUI)
*** Creación de Herramientas de Línea de Comandos
  
```

```
** Desarrollo de Juegos con Python
*** Introducción a Pygame
*** Física y Animación de Juegos
*** Patrones de Diseño de Juegos
*** Creación de Recursos para Juegos (gráficos, audio)
*** Desarrollo de Juegos Multijugador
*** Distribución y Despliegue de Juegos
*** Conceptos Avanzados de Desarrollo de Juegos

** DevOps con Python
*** Introducción a los Principios de DevOps
*** Infraestructura como Código (usando herramientas como Ansible o Terraform)
*** Integración y Despliegue Continuo (CI/CD)
*** Gestión de Configuración (usando herramientas como Puppet o Chef)
*** Monitoreo y Registro de Transacciones
*** Contenerización (usando Docker)
*** Plataformas en la Nube (AWS, Azure, GCP)

** Conceptos Avanzados de Python
*** Decoradores
*** Generadores e Iteradores
*** Gestores de Contexto
*** Metaprogramación
*** Concurrencia y Paralelismo
*** Patrones de Diseño en Python
*** Optimización de Código Python

** Pruebas y Depuración en Python
*** Pruebas Unitarias (usando marcos como pytest)
*** Desarrollo Dirigido por Pruebas (TDD)
*** Técnicas de Depuración
*** Perfilado y Optimización de Rendimiento de Código
*** Manejo de Errores y Excepciones
*** Escritura de Código Mantenible y Probable
*** Revisión de Código e Integración Continua

** Recursos Adicionales
*** Cursos y Tutoriales en Línea
*** Libros y Libros Electrónicos
*** Desafíos de Codificación y Plataformas de Práctica
*** Proyectos y Contribuciones de Código Abierto
*** Comunidad y Foros de Python
*** Encuentros y Conferencias de Programadores Python
@endmindmap
```

7.2 Lecturas sugeridas

1. Python.org. (n.d.). Python documentation. <https://www.python.org/doc/>
2. Boulouma, A. (2023). A Comprehensive Guide to Software Development effortlessly with Python Programming. <https://aminblm.gumroad.com/l/python>
3. Downey, A. (2015). Think Python: How to Think Like a Computer Scientist. O'Reilly Media, Inc. <https://greenteapress.com/wp/think-python-2e/>
4. Sweigart, A. (2019). Automate the Boring Stuff with Python: Practical Programming for Total Beginners. No Starch Press. <https://automatetheboringstuff.com/>
5. GeeksforGeeks. (2023). Python Tutorial. <https://www.geeksforgeeks.org/python-programming>
6. Real Python (n.d). <https://realpython.com/>
7. Parlante, N. (2022). Google's Python Class. <https://developers.google.com/edu/python/>

Fin

Lista de los 100 *Prompts*

1. “ChatGPT: Tu Entrenador de Python. Aprende los Fundamentos en 100 Prompts” es el título de un libro que estoy escribiendo para enseñar los fundamentos de Python. Proporciona una versión preliminar de un prólogo atractivo y provocador. Enfatiza el impacto que la IA, y los modelos de lenguaje en particular, están teniendo en muchas áreas, incluida la programación, y cómo se espera que aprovechen las habilidades y la productividad de los programadores para crear aplicaciones poderosas en un futuro cercano, mencionando de pasada que los problemas relacionados con el contenido falso y la generación de información errónea que se han detectado en estos modelos de lenguaje se alivian en el área de la programación porque el código generado se puede validar simplemente ejecutándolo. También comenta que el contenido generado inicialmente ha sido cuidadosamente revisado y enriquecido basado en mi experiencia educativa.
2. Explica con un tono colorido que el propósito de este libro es introducir a un principiante en el mundo de la programación, en particular en el lenguaje Python, y ayudarlo a aprender los conceptos básicos con ejemplos, explicaciones y retroalimentación, al igual que un profesor guía a un estudiante en su viaje de aprendizaje. Solo que aquí, ese profesor es ChatGPT. Enfatiza las nuevas posibilidades y ventajas de utilizar desarrollos modernos en IA para ayudar en el proceso de aprendizaje y desarrollo de programadores.
3. Proporciona una descripción ingeniosa del lenguaje de programación Python que sea adecuada para un estudiante universitario de primer año. Haga algunas notas sobre la historia del lenguaje y su versión actual, especula sobre su futuro y sugiere enlaces útiles a organizaciones o sitios web donde el lector pueda obtener más información técnica.
4. Escribe una breve nota sobre Monty Python.
5. Describe brevemente las diferencias clave entre las versiones Python 2.x y Python 3.x. Destaca las ventajas de la versión más reciente.
6. Crea un esquema del ciclo estándar de desarrollo de programas, utilizando el paquete LaTeX smartdiagram con el tipo de diagrama circular.
7. Explica a un lector universitario de primer año el ciclo estándar de desarrollo de programas, incluyendo los pasos de escribir (codificar), ejecutar, probar y corregir (depurar). Hazlo con una lista con viñetas.
8. Aclara que Python es un lenguaje de programación interpretado; explica cómo difiere de un lenguaje compilado (ventajas y desventajas). Menciona que hay una variedad de IDEs disponibles, pero en este libro, en lugar de lidiar con problemas de instalación de software, utilizaremos entornos listos para usar que satisfacen las necesidades de este libro, para que podamos centrarnos en conceptos y algoritmos. Se utilizarán Jupiter Notebooks, Colaboratory, PythonTutor y ChatGPT como herramientas. Haz una lista con viñetas de estas herramientas, incluyendo una breve descripción, el sitio web y los pasos necesarios para abrir una cuenta si se requiere.
9. Describe detalladamente cómo crear un nuevo cuaderno Jupyter en Google Colab.
10. Explica los dos tipos de celdas en Jupyter Notebook.
11. Explica en detalle cómo usar PythonTutor.

12. Explícale a un estudiante universitario de primer año qué es ChatGPT, cómo puede ayudar a los estudiantes a aprender programación, cómo utilizarlo de manera responsable para lograr buenos objetivos de aprendizaje, si existen preocupaciones éticas sobre su uso o dependencia excesiva y cómo abordarlas.
13. Sugiere plantillas de preguntas sobre cómo un estudiante de programación puede preguntarle a ChatGPT para obtener una respuesta precisa y atractiva sobre los siguientes temas: 1) Presentar y explicar conceptos de programación de manera accesible. 2) Crear ejemplos de código relacionados con esos conceptos. 3) Describir las líneas clave de dicho código. 4) Escribir documentación y comentarios en el código. 5) Basado en el ejemplo, proponer nuevos ejercicios para que el estudiante practique variaciones del problema resuelto. 6) Incluir enlaces útiles a recursos que complementen los temas anteriores. 7) Hacer comentarios divertidos e ingeniosos para hacer el tema más agradable.
14. Explica el concepto del programa “¡Hola, mundo!” de una manera sencilla pero colorida, sin utilizar código. Además, incluye una anécdota relacionada para proporcionar a los principiantes en programación una introducción cálida al tema.
15. Pero, ¿por qué el mensaje ¡Hola, mundo!? ¿Quién lo inventó? ¿Cuál es su significado?
16. Escribe el clásico programa ‘¡Hola, mundo!’ en Python y explícalo. Proporciona también la salida. Utilizaré Colab.
17. Explica qué es un error de sintaxis y muestra 3 posibles errores de sintaxis en el programa ‘¡Hola, mundo!’ con sus respectivas salidas.
18. ¿Por qué se llama print()? ¿Necesito una impresora para poder usarlo?
19. Explica de manera sencilla qué es una cadena de caracteres y las reglas de sintaxis para definir cadenas en Python.
20. Explica el formateo de cadenas de caracteres de manera sencilla. Escribe ejemplos de código divertidos para cada una de las opciones de formato
21. ¿Por qué hay algunas líneas verdes en el código que comienzan con un símbolo de numeral?
22. Ahora dame un ejemplo de un programa que use solo comandos print() con una analogía del mundo real.
23. ¿Puedo utilizar opciones de formato como ancho y alineación para construir cadenas? Muéstrame ejemplos interesantes.
24. ¿Qué se entiende por número “de punto flotante”?
25. Me gustaría poner en práctica lo que acabo de aprender sobre print() y el formateo de cadenas. Por favor, proporciona cuatro ejercicios de código en los que deba corregir los errores para que se ejecuten correctamente.
26. Me gustaría poner en práctica lo que acabo de aprender sobre print() y el formateo de cadenas. Esta vez, sugiere algunos ejercicios que se puedan resolver utilizando solo print() y el formateo de cadenas.
27. Menciona detalladamente que una característica interesante de ChatGPT es que puedes pedirle que edite un fragmento de código, lo corrija, lo mejore o proporcione comentarios.

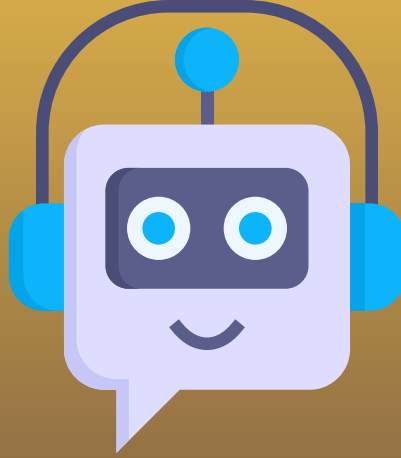
28. Crea una plantilla de prompt simple para revisar/mejorar/corregir un código dado.
29. A continuación se muestra el código que escribí para resolver el ejercicio: “Crea un programa que imprima un triángulo rectángulo utilizando el carácter ‘*’ “. Por favor, revisa/mejora/corriges mi código y proporciona comentarios sobre cómo mejorarlo, hacerlo más eficiente o más legible. Además, demuestra cómo habrías codificado la solución de manera diferente, si corresponde. Código: `print('* ') print('* ' * 2) print('* ' * 3) print('* ' * 4) print('* ' * 5)`
30. ¿Qué es exactamente un programa? ¿Cómo se relaciona con los datos?
31. Explica con ejemplos y salidas en pantalla cómo se representan los datos en Python. Proporciona ejemplos de personajes de cómics para entender mejor el concepto.
32. Entonces, ¿cómo puedo saber el tipo de una variable en un punto dado del programa?
33. ¿Cómo puede un programa recibir entrada del usuario de diferentes tipos de datos? Utiliza ejemplos de cómics.
34. ¿Qué significan esas sentencias if/else que utilizaste en algunos de los ejemplos anteriores? Puedo entender la idea básica, pero ¿puedes proporcionar una explicación clara y concisa?
35. ¡Hola, ChatGPT! Me gustaría practicar mis habilidades de programación con un ejercicio que involucre entrada de usuario, conversión de tipos de datos e impresión de salida, tal vez incluso incluyendo un menú de opciones. ¿Podrías sugerirme un ejercicio en el que pueda trabajar? Una vez que tenga el enunciado, también agradecería si me mostraras un ejemplo de la salida esperada. ¡Gracias!
36. Bien, ahora muéstrame la solución.
37. Genial. Mi versión de la solución se muestra a continuación. ¿Qué piensas al respecto? ¿En comparación con la tuya?
38. Pero me aseguré de ocultar el número de tarjeta de crédito. Solo estaba bromeando.
39. Por cierto, me gusta cómo comentaste cada sección de tu código; ayuda a aclarar las cosas. Por favor, haz lo mismo con mi código.
40. Proporciona cuatro ejercicios de código relacionados con la entrada y conversión de tipos de datos que necesito corregir para que se ejecuten correctamente, ajustando errores de sintaxis o lógica.
41. Sugiere algunos ejercicios que requieran escribir código para resolverlos, utilizando solo la entrada del usuario y operaciones de conversión de tipos de datos.
42. Enumera todos los operadores de Python y proporciona una breve descripción para cada uno.
43. Explica los detalles de los operadores aritméticos en Python en un tono divertido e incluye un ejemplo de código para ilustrar el uso de todos los operadores. El ejemplo debe comparar estadísticas entretenidas sobre las películas de Marvel y X-Men y su recaudo en taquilla.
44. ¿Qué significan esos números 22.55e9 y 6.01e9 para las variables de taquilla?

45. Explica los detalles de los operadores de comparación en Python con un tono divertido e incluye un ejemplo de código para ilustrar el uso de todos ellos, esta vez utilizando datos curiosos sobre equipos de fútbol.
46. Explica el concepto de los operadores lógicos en Python en un tono informal. Después de definir cada operador, incluye un ejemplo de código para ilustrar el uso de todos ellos. Para este fin, usa una tabla de platos de comida rápida indicando su idoneidad para vegetarianos y veganos.
47. Explica el concepto de los operadores a nivel de bit en Python de forma amigable pero profesional. Incluye un ejemplo de código después de definir cada operador para demostrar cómo usarlos. Muestra los resultados de las operaciones, incluyendo las secuencias de bits de los operandos y sus correspondientes representaciones en decimal.
48. De acuerdo, pero ¿podrías explicar brevemente a qué te refieres con “dígitos binarios de un entero”? Explícalo con algunas analogías de la vida real que sean divertidas.
49. De acuerdo, una última cosa: por favor, explícame qué quieres decir con “representación en forma de complemento a dos” de manera breve y amigable.
50. Explica el concepto de los operadores de asignación aritmética en Python de manera divertida.
51. Explica el concepto de los operadores de identidad en Python de manera divertida.
52. Vale. Dos preguntas. Primero, ¿a qué te refieres exactamente con “ubicaciones en la memoria”? No veo ninguna ubicación de memoria en el código; por favor, acláralo de manera amigable y colorida. Segundo, ¿qué es la variable especial de Python “None”? Por favor, acláralo.
53. Explica el concepto de los operadores de pertenencia en Python de manera lúdica.
54. Explica el concepto del operador de exponenciación en Python de manera lúdica.
55. ¿Qué sucede cuando una expresión tiene múltiples operandos? ¿En qué orden se evalúan?
56. ¿Cómo puedo programar operaciones matemáticas más complejas como raíces cuadradas, logaritmos, y otros?
57. Sugiere algunos ejercicios divertidos e informativos para practicar los operadores aritméticos, de comparación, lógicos, a nivel de bits, de pertenencia, de exponenciación y matemáticos.
58. Ahora sugiere un problema de codificación interesante y atractivo para practicar la representación binaria de números decimales. Asume que tienes que mostrar una tabla de platos de comida, con columnas que representan los ingredientes utilizados para prepararlos y una columna final que muestra el código decimal del plato como el resultado de interpretar las columnas de ingredientes como bits de un número binario (muestra un ejemplo de una tabla con cuatro platos y cinco ingredientes). El programa debe solicitar el nombre de un plato y una serie de preguntas tipo sí/no para determinar el código decimal del plato, y luego imprimir una fila de la tabla con esta información (muestra un ejemplo de la salida en pantalla de la fila resultante).
59. Enumera todas las estructuras de control en programación y proporciona una breve descripción de cada una.

60. Explícame todo lo que necesito saber sobre las sentencias if/else en Python de una manera divertida y creativa, con ejemplos juguetones.
61. ¿Hay algo más que deba saber sobre las estructuras condicionales en Python?
62. Ahora hablemos sobre las sentencias match/case en Python. Explícame todo lo que necesito saber sobre ellas de una manera creativa, utilizando ejemplos divertidos.
63. De acuerdo. En la sentencia match, veo que usaste el comando "lower()". ¿Debo usarlo cada vez que codifique una sentencia match/case?
64. Por favor, proporciona una breve definición técnica del bucle for.
65. Ahora veamos los bucles for en acción de manera divertida y creativa, utilizando ejemplos lúdicos.
66. Proporciona una breve definición técnica del bucle while.
67. Ahora veamos los bucles while en acción de una manera divertida y creativa, utilizando ejemplos lúdicos.
68. En el ejemplo "Contando ovejas", preguntas al usuario si está dormido dos veces. ¿No es redundante? ¿Hay alguna forma de escribir el bucle sin hacer la misma pregunta dos veces?
69. ¿Hay algo más que deba saber sobre los bucles for o while en Python?
70. ¿Qué es una estructura de control anidada y por qué es útil?
71. Ahora adentrémonos en las estructuras de control anidadas en Python. Muéstrame cómo usar condicionales anidados permitiendo múltiples niveles de ramificación, bucles anidados incluyendo bucles for dobles, bucles while dobles y bucles mixtos while/for o for/while, así como combinaciones de bucles dentro de condicionales y condicionales dentro de bucles. Utiliza ejemplos de superhéroes y cómics para ser creativo y lúdico en tu explicación.
72. Sugiere algunos ejercicios de codificación divertidos y atractivos para practicar las sentencias if/else, if/elif/else y match/case, así como el operador ternario, incluyendo descripciones de problemas y ejemplos de salidas esperadas.
73. Sugiere algunos ejercicios de codificación divertidos y atractivos para practicar bucles for y while, incluyendo descripciones de problemas y ejemplos de salidas esperadas.
74. Sugiere algunos ejercicios de programación divertidos e interesantes para practicar estructuras de bucles anidados, incluyendo descripciones de problemas y ejemplos de resultados esperados.
75. Sugiere el problema de escribir un programa de calculadora basado en consola que tome dos números y una operación seleccionada de un menú (decorado con un marco), calcule y muestre el resultado, y pregunte al usuario si quiere continuar. Incluye un ejemplo de la salida esperada.
76. Explica qué es una estructura de contenedor, luego enumera todas las estructuras de contenedor disponibles en Python y proporciona una breve descripción para cada una, utilizando analogías de la vida real y un tono amigable.

77. Explica todo lo que necesito saber sobre listas en Python de una manera divertida y creativa, usando ejemplos lúdicos.
78. De acuerdo. Muéstrame un ejemplo de código que ilustre cada uno de los elementos mencionados anteriormente. Incluye comentarios en cada bloque de código para explicar su propósito. Utiliza si quieres el tema de superhéroes contra villanos.
79. Interesante. Ahora demuéstreme cómo puedo usar bucles y condicionales con listas. Esta vez, utiliza ejemplos creativos relacionados con comida, aprovechando las características de aleatoriedad de Python.
80. ¡Espera un momento! En los ejemplos anteriores, usas algunos trucos interesantes de los que no estoy al tanto, como `random.choice()`, `enumerate()` e `insert()`. ¿Podrías explicármelos, por favor?
81. Explícame todo lo que necesito saber sobre las tuplas en Python de una manera divertida y creativa, usando ejemplos lúdicos.
82. Ok. .Ahora veamos algunos ejemplos de código para ver las tuplas en acción. ¿Por qué no usamos bandas de rock esta vez para ilustrar los elementos que mencionaste anteriormente?
83. ¡Espera! Hay un asterisco (*) extraño antes de la variable “integrantes” en el ejemplo del truco de desempaquetado. ¿Qué significa eso exactamente?
84. Explícame todo lo que necesito saber sobre los conjuntos en Python de una manera divertida y creativa, usando ejemplos lúdicos.
85. Muy bien. Ahora veamos algunos ejemplos de código para ver los conjuntos en acción. Esta vez, usemos datos divertidos sobre los planetas para ilustrar los elementos que mencionaste anteriormente.
86. Explícame todo lo que necesito saber sobre las cadenas de caracteres en Python de una manera divertida y creativa, usando ejemplos lúdicos.
87. ¡Que bien! Ahora veamos algunos ejemplos de código para ver las cadenas de texto en acción! Considera usar libros clásicos infantiles para ilustrar los elementos que mencionaste anteriormente.
88. Explícame todo lo que necesito saber sobre los diccionarios en Python de una manera divertida y creativa, utilizando ejemplos lúdicos.
89. ¡Genial! ¡Ahora veamos algunos ejemplos de código para ver los diccionarios en acción! Mantengámonos en el tema de los superhéroes para ilustrar los elementos que mencionaste anteriormente.
90. ¿Qué tal crear una enciclopedia de superhéroes? ¿Puedo implementarla como un diccionario de diccionarios? ¿Alguna especie de diccionario anidado? Ilústralo con un ejemplo de código.
91. ¡Súper! También mencionaste que los diccionarios nos permiten explorar relaciones y conexiones complejas entre los elementos dentro del diccionario. ¿Qué significa eso? ¿Pueden los diccionarios ayudarme a determinar relaciones como “los enemigos de mis aliados son mis enemigos”, “los aliados de mis aliados son mis amigos”, “los héroes con el mismo superpoder que el mío son mis amigos” o “mostrar una tabla de superhéroes ordenados por año de estreno”? Por favor, proporciona algunos ejemplos.
92. Sugiere algunos ejercicios divertidos e interesantes para practicar con listas, tuplas y conjuntos. Proporciona una descripción del problema y ejemplos de los resultados esperados.

93. Sugiere algunos ejercicios divertidos e interesantes para practicar con cadenas y diccionarios, incluidos diccionarios anidados. Proporciona una descripción del problema y ejemplos de los resultados esperados.
94. Explica brevemente el concepto de modularización y reusabilidad, así como por qué son útiles en proyectos de programación. Luego, utilizando analogías del mundo real, explica los diversos enfoques para implementar estas ideas de manera clara y amigable.
95. ¡Uff! Que impresionante lista de ideas poderosas. Sin embargo, como aún estoy aprendiendo, por favor, recomiéndame uno o dos elementos de la lista con los que debería comenzar como principiante, y luego describe los componentes clave de cada elemento de manera relajada y entretenida.
96. ¡Vale! ¡Comencemos con algunos ejemplos de código para ver las funciones en acción! Considera usar ideas relacionadas con redes sociales para ilustrar los elementos que enumeraste anteriormente. Podrías usar datos curiosos de las historias de Harry Potter, así como funcionalidades aleatorias disponibles en Python.
97. ¡Muy bien! ¡Continuemos con algunos ejemplos de código para ver las clases en acción! Considera usar preguntas de películas y personajes clásicos para ilustrar los elementos que mencionaste anteriormente, incluyendo funcionalidades aleatorias de Python.
98. Sugiere algunos ejercicios de programación divertidos y atractivos para practicar funciones, valores de retorno, parámetros y expresiones lambda. Cada ejercicio debe ir acompañado de una descripción del problema y ejemplos de la salida esperada. Puedes incorporar elementos de matemáticas o aleatoriedad, que requieran importar librerías necesarias para resolver los problemas.
99. ¡Vale! Ahora sugiere algunos ejercicios de programación divertidos y atractivos para practicar POO, incluyendo clases, constructores, atributos y métodos. Cada ejercicio debe ir acompañado de una descripción del problema y ejemplos de la salida esperada. Puedes incorporar elementos de matemáticas o aleatoriedad, que requieran importar librerías necesarias para resolver los problemas.
100. ¡Ha sido un emocionante viaje hasta hasta el momento! Pero ahora que he dominado los fundamentos de la programación en Python (tipos de datos, variables, entrada/salida de datos por consola, operadores, condicionales, bucles, listas, tuplas, cadenas, diccionarios, funciones, clases y objetos), ¿hacia adonde debería continuar? ¿Podrías guiarme en la dirección correcta? Sugiere algunos planes de estudio para profundizar en diferentes rutas de aprendizaje que me ayuden a convertirme en un experto de la programación en Python. (usa un formato de mapa mental como PlantUML)



Esta guía ha sido cuidadosamente diseñada para ayudarte a navegar los conceptos básicos de la programación en Python con facilidad, usando ChatGPT como tu confiable entrenador.

ChatGPT es un potente modelo de lenguaje capaz de ayudarte a resolver preguntas o dudas que surjan mientras aprendes Python.

Te proporcionará ejemplos, explicaciones y retroalimentación, haciendo que tu experiencia de aprendizaje sea interactiva y estimulante.