

Guía Práctica Rápida Cómo Programar y Aprender Python con ChatGPT

Juan Monroy



Una de las últimas revoluciones tecnológicas, como sabrás, son los modelos de lenguaje GPT.

En concreto, el ya archiconocido ChatGPT es capaz de entender y generar código Python.

Lejos de entrar en el debate de si la Inteligencia Artificial acabará con el trabajo de los desarrolladores (cosa que me parece improbable), en esta guía quiero que tengas una introducción rápida y práctica para usar ChatGPT como un asistente de programación que puede ahorrarte mucho trabajo y aumentar tu productividad.

Pero antes ten en cuenta el consejo fundamental: tienes que ser crítico con las respuestas de ChatGPT, debes entenderlas bien y para eso tienes que entender el lenguaje.

De nada sirve pedirle código a ChatGPT que no entiendas pues puede contener errores o, simplemente, ser muy mejorable.

Entiende esto como lo que es: una herramienta de trabajo más y, como tal, hay que saber usarla y comprender tanto sus funcionalidades como sus límites.

Contenidos:

Creación de funciones y scripts	2
Refactorización	4
Estudiar y practicar exámenes	6
Evitar bloqueos	7
Trabajo tedioso	9
Explicación de conceptos	11
Creación de ficheros de entrada de prueba	13
Tests	15
Intérprete de Python	17
Asesor de tecnología y librerías	19

Creación de funciones y scripts

El uso más evidente que podemos darle a ChatGPT es el de la escritura de código.

Podemos pedirle que escriba funciones o scripts de lo que necesitemos.

Aquí van algunos ejemplos:

Prompt:

Escribe una función en Python que reciba tres números como parámetro y nos devuelva el mayor de ellos.

Respuesta:

¡Claro! Aquí te dejo una función en Python que recibe tres números como parámetros y devuelve el mayor de ellos:

```
def encontrar_mayor(num1, num2, num3):  
    if num1 >= num2 and num1 >= num3:  
        return num1  
    elif num2 >= num1 and num2 >= num3:  
        return num2  
    else:  
        return num3
```

Esta función utiliza una serie de condicionales para determinar cuál de los tres números es el mayor. Primero compara `num1` con `num2` y `num3`. Si `num1` es mayor o igual que ambos, entonces devuelve `num1` como el número mayor. De lo contrario, compara `num2` con `num1` y `num3`. Si `num2` es mayor o igual que ambos, entonces devuelve `num2`. Si ninguna de estas condiciones es verdadera, entonces `num3` debe ser el mayor, por lo que se devuelve `num3`.

Puedes llamar a esta función con tus propios valores numéricos de la siguiente manera:

```
print(encontrar_mayor(4, 7, 2)) # Debería imprimir 7  
print(encontrar_mayor(-3, -8, -1)) # Debería imprimir -1  
print(encontrar_mayor(0, 0, 0)) # Debería imprimir 0
```

Espero que te sea útil!

Prompt:

Escribe un script de Python que pida al usuario 3 números por teclado, compruebe que dichos números estén entre 0 y 100 y utilice la función que acabas de crear para mostrar por pantalla el mayor de ellos. La comprobación debe estar implementada en una función. Si no se cumplen las condiciones, los números se vuelven a pedir al usuario.

Resultado:

¡Por supuesto! Aquí tienes un script de Python que cumple con tus especificaciones:

```
def comprobar_numero(num):
    if num < 0 or num > 100:
        return False
    else:
        return True

while True:
    num1 = int(input("Introduce el primer número (entre 0 y 100): "))
    if comprobar_numero(num1):
        break
    else:
        print("El número debe estar entre 0 y 100. Inténtalo de nuevo.")

while True:
    num2 = int(input("Introduce el segundo número (entre 0 y 100): "))
    if comprobar_numero(num2):
        break
    else:
        print("El número debe estar entre 0 y 100. Inténtalo de nuevo.")

while True:
    num3 = int(input("Introduce el tercer número (entre 0 y 100): "))
    if comprobar_numero(num3):
        break
    else:
        print("El número debe estar entre 0 y 100. Inténtalo de nuevo.")

mayor = encontrar_mayor(num1, num2, num3)
print("El número mayor es:", mayor)
```

Este script utiliza la función `comprobar_numero` para verificar si los números introducidos están entre 0 y 100. Si un número está fuera de ese rango, el script volverá a pedir al usuario que lo introduzca. Una vez que se han introducido tres números válidos, se utiliza la función `encontrar_mayor` para determinar cuál es el número mayor y se muestra por pantalla.

Refactorización

También podemos usarlos para mejorar o refactorizar código tanto nuestro como generado por el propio ChatGPT.

En el ejemplo anterior hay algunas cosas mejorables, como la función de comprobación y la repetición de código.

Vamos a pedirle que lo mejore.

Prompt:

¿Puedes refactorizar ese último código para evitar repeticiones y mejorar la comprobación (el if no es necesario)?

Resultado:

¡Por supuesto! Aquí está el código refactorizado:

```
def comprobar_numero(num):  
    return num >= 0 and num <= 100  
  
numeros = []  
  
for i in range(3):  
    while True:  
        num = int(input(f"Introduce el {i+1}º número (entre 0 y 100): "))  
        if comprobar_numero(num):  
            numeros.append(num)  
            break  
        else:  
            print("El número debe estar entre 0 y 100. Inténtalo de nuevo.")  
  
mayor = encontrar_mayor(*numeros)  
print("El número mayor es:", mayor)
```

En este código, la función `comprobar_numero` ha sido simplificada para devolver `True` si el número está entre 0 y 100, y `False` de lo contrario. Luego, se utiliza un bucle `for` para pedir tres números al usuario. En cada iteración del bucle, se comprueba si el número introducido es válido utilizando la función `comprobar_numero`. Si es así, se agrega a la lista `numeros`. Si no, se le pide al usuario que lo vuelva a introducir.

Finalmente, se llama a la función `encontrar_mayor` con la lista `numeros` como argumento y se muestra el número mayor por pantalla.

Espero que esto te resulte útil y te ayude a mejorar tu código!

Ahora veremos otro ejemplo con un pequeño código nuestro a ver si es capaz de ver la posible mejora:

Prompt:

Tengo el siguiente código en Python:

```
num1, num2, num3, num4 = 10, 20, 30, 40
resultado1 = (num1 + num2) / (num1 * num2)
resultado2 = (num3 + num4) / (num3 * num4)
print(f'{resultado1=} {resultado2=}')

```

¿Puedes refactorizarlo usando funciones?

Resultado:

¡Por supuesto! Aquí te dejo una posible versión refactorizada del código utilizando funciones:

```
def calcular_resultado(num1, num2):
    """
    Esta función calcula el resultado de la fórmula (num1 + num2) / (num1 *
num2)
    """
    return (num1 + num2) / (num1 * num2)

# Definimos las variables num1, num2, num3 y num4
num1, num2, num3, num4 = 10, 20, 30, 40

# Calculamos los resultados utilizando la función calcular_resultado
resultado1 = calcular_resultado(num1, num2)
resultado2 = calcular_resultado(num3, num4)

# Mostramos los resultados por pantalla
print(f'{resultado1=} {resultado2=}')

```

En esta versión refactorizada del código, hemos creado una función `calcular_resultado` que toma como parámetros `num1` y `num2` y devuelve el resultado de la fórmula $(\text{num1} + \text{num2}) / (\text{num1} * \text{num2})$. De esta forma, podemos utilizar esta función para calcular los resultados de la fórmula para las variables `num1` y `num2`, y para las variables `num3` y `num4`.

En lugar de calcular los resultados de la fórmula directamente en el código, ahora los calculamos utilizando la función `calcular_resultado`. Esto hace que el código sea más modular y fácil de entender, y nos permite reutilizar la función `calcular_resultado` en otros lugares del programa si es necesario.

Estudiar y practicar exámenes

¿Y por qué no usar a ChatGPT como un profesor de programación? ¡Cuántas posibilidades!

Prompt:

Necesito que te comportes como un profesor de programación. ¿Puedes hacerme un examen tipo test de Python acerca de las listas? El examen debe constar de 3 preguntas. Las preguntas deben de tener 4 posibles respuestas. Después tendré que darte la solución al examen y tú deberás decirme cuáles he acertado y cuáles he fallado.

Resultado:

¡Por supuesto! A continuación te presento un examen de 3 preguntas acerca de listas en Python con 4 opciones de respuesta cada una:

1. ¿Qué función se utiliza para añadir un elemento al final de una lista en Python?
 - a) `append()`
 - b) `add()`
 - c) `extend()`
 - d) `insert()`
2. ¿Cuál es el índice del último elemento de una lista en Python?
 - a) `len(lista) - 1`
 - b) `len(lista)`
 - c) `lista[len(lista) - 1]`

d) lista[len(lista)]

3. ¿Qué función se utiliza para ordenar una lista de forma ascendente en Python?

a) sort()

b) sorted()

c) order()

d) arrange()

Para comprobar tus respuestas, por favor indícame qué opción seleccionaste para cada pregunta.

Prompt:

1. a

2. a

3. d

Resultado:

¡Muy bien! Has acertado en las preguntas 1 y 2, pero has fallado en la pregunta 3. La respuesta correcta es la opción a), `sort()`. La función `sort()` ordena una lista de forma ascendente in-place, es decir, modifica la lista original. Por otro lado, `sorted()` devuelve una nueva lista ordenada sin modificar la lista original. Las opciones c) y d) no existen en Python. ¡Gracias por realizar el examen!

Evitar bloqueos

A veces tenemos que resolver un problema y no sabemos ni cómo empezar.

En lugar de buscar en Google o StackOverflow podemos arrancar consultándole a ChatGPT.

Prompt:

¿Qué tipos de algoritmos existen para ordenar una lista en Python? Me interesan los más eficientes.

Resultado:

Existen varios tipos de algoritmos para ordenar una lista en Python. Aquí te menciono algunos de los más eficientes:

1. Quicksort: Es uno de los algoritmos de ordenamiento más utilizados en la práctica, ya que tiene un tiempo de ejecución promedio de $O(n \log n)$. Es un algoritmo recursivo que divide la lista en sub-listas más pequeñas y las ordena por separado.
2. Mergesort: Es otro algoritmo de ordenamiento eficiente que tiene un tiempo de ejecución promedio de $O(n \log n)$. Este algoritmo también divide la lista en sub-listas más pequeñas y las ordena por separado. Luego, combina las sub-listas ordenadas para obtener la lista ordenada completa.
3. Heapsort: Es un algoritmo de ordenamiento eficiente que tiene un tiempo de ejecución promedio de $O(n \log n)$. Este algoritmo utiliza una estructura de datos llamada heap para ordenar la lista.
4. Timsort: Es un algoritmo de ordenamiento híbrido que utiliza una combinación de mergesort y heapsort. Este algoritmo tiene un tiempo de ejecución promedio de $O(n \log n)$ y se utiliza en la implementación de la función `sorted()` en Python.

Cada uno de estos algoritmos tiene ventajas y desventajas dependiendo del tipo de datos que se van a ordenar. En general, Quicksort y Mergesort son los algoritmos más utilizados en la práctica debido a su tiempo de ejecución promedio de $O(n \log n)$.

Prompt:

Tengo que leer datos de un fichero en Python y no sé ni cómo empezar. ¿Alguna sugerencia?

Resultado:

Por supuesto, leer datos de un archivo en Python es una tarea muy común y sencilla. Puedes hacerlo de varias maneras, pero aquí te muestro una forma simple:

1. Abre el archivo utilizando la función `open()` y guarda el objeto de archivo en una variable.
2. Lee el contenido del archivo utilizando el método `read()`.
3. Cierra el archivo utilizando el método `close()`.

Aquí te muestro un ejemplo básico para leer el contenido de un archivo de texto:

```
# Abrir el archivo en modo lectura
archivo = open('archivo.txt', 'r')
```

```
# Leer el contenido del archivo
contenido = archivo.read()
```

```
# Cerrar el archivo
archivo.close()
```



```
# Imprimir el contenido del archivo
print(contenido)
```

En este ejemplo, `'archivo.txt'` es el nombre del archivo que deseas leer. Asegúrate de que el archivo exista en el directorio actual. El modo `'r'` significa que se abrirá el archivo en modo lectura.

Ten en cuenta que si el archivo es muy grande, leerlo todo de una sola vez puede no ser la mejor opción. En ese caso, podrías leerlo por partes utilizando un bucle `for` o el método `readline()`. Además, también existen módulos específicos para leer archivos de diferentes formatos, como CSV, JSON, entre otros.

Trabajo tedioso

Si necesitamos escribir un script o función sencillos de describir con lenguaje natural, pero tediosos de implementar, ChatGPT será un gran aliado.

Prompt:

Necesito un diccionario de Python que asocie cada número del 1 al 20 su nombre.

Resultado:

Por supuesto, aquí tienes un diccionario que asocia cada número del 1 al 20 con su nombre:

```
numeros = {
    1: 'uno',
    2: 'dos',
    3: 'tres',
    4: 'cuatro',
    5: 'cinco',
    6: 'seis',
    7: 'siete',
    8: 'ocho',
    9: 'nueve',
    10: 'diez',
    11: 'once',
    12: 'doce',
    13: 'trece',
    14: 'catorce',
```

```
15: 'quince',
16: 'dieciséis',
17: 'diecisiete',
18: 'dieciocho',
19: 'diecinueve',
20: 'veinte'
}
```

Puedes acceder al nombre de un número específico usando su clave, por ejemplo, `numeros[5]` devuelve `'cinco'`.

Prompt:

Necesito una clase de Python llamada Contacto y que tenga los siguientes atributos: nombre (string), apellidos(string), edad (número entero). Crea properties para poder acceder y modificar los atributos. Debe tener una función que calcule el año de nacimiento aproximado del contacto. Usa type hinting.

Resultado:

Claro, aquí tienes una posible implementación de la clase `Contacto` con los atributos `nombre`, `apellidos` y `edad` como propiedades:

```
from datetime import date

class Contacto:
    def __init__(self, nombre: str, apellidos: str, edad: int):
        self._nombre = nombre
        self._apellidos = apellidos
        self._edad = edad

    @property
    def nombre(self) -> str:
        return self._nombre

    @nombre.setter
    def nombre(self, value: str):
        self._nombre = value

    @property
    def apellidos(self) -> str:
        return self._apellidos

    @apellidos.setter
    def apellidos(self, value: str):
```

```
        self._apellidos = value

@property
def edad(self) -> int:
    return self._edad

@edad.setter
def edad(self, value: int):
    self._edad = value

def calcular_anio_nacimiento(self) -> int:
    anio_actual = date.today().year
    anio_nacimiento = anio_actual - self.edad
    return anio_nacimiento
```

En esta implementación, los atributos `nombre`, `apellidos` y `edad` se han convertido en propiedades mediante los decoradores `@property` y `@atributo.setter`. Estos decoradores permiten acceder y modificar los atributos de la clase a través de métodos que se comportan como atributos.

Además, se ha utilizado type hinting para especificar el tipo de los argumentos y el tipo de retorno de los métodos.

Para crear una instancia de la clase `Contacto`, puedes hacer lo siguiente:

```
contacto = Contacto("Juan", "Pérez González", 30)
print(contacto.nombre) # Juan
print(contacto.apellidos) # Pérez González
print(contacto.edad) # 30
contacto.edad = 35
print(contacto.edad) # 35
print(contacto.calcular_anio_nacimiento()) # 1988 (aproximadamente)
```

Explicación de conceptos

Si no entendemos alguna cosa concreta podemos pedirle que nos la explique de manera sencilla.

Por ejemplo, vamos a pedirle que nos explique dos conceptos concretos de Python que son los generadores y la gestión de contexto.

Prompt:

¿Puedes explicarme de manera sencilla qué son los generadores de Python?

Resultado:

¡Por supuesto! En Python, un generador es una función que utiliza la palabra clave `yield` para devolver valores de manera iterativa, en lugar de utilizar la palabra clave `return` para devolver un valor de forma definitiva y terminar la ejecución de la función.

Cuando una función utiliza `yield` en lugar de `return`, Python entiende que esta función es un generador. Al llamar a un generador, éste devuelve un objeto iterable que podemos recorrer para obtener los valores que devuelve la función mediante la palabra clave `yield`.

Cada vez que se llama al generador, la función se ejecuta hasta que encuentra la siguiente instrucción `yield`. Entonces, la función devuelve el valor asociado a ese `yield` y se detiene en ese punto. La próxima vez que se llame al generador, la función continúa su ejecución justo después del último `yield`, y así sucesivamente hasta que ya no hay más valores que devolver.

En resumen, los generadores en Python nos permiten devolver valores de manera iterativa sin tener que almacenar todos los valores en una lista o en otro objeto iterable. Esto puede ser útil cuando trabajamos con grandes cantidades de datos o cuando no sabemos de antemano cuántos valores vamos a devolver, ya que los generadores nos permiten devolver valores uno a uno, de forma eficiente y sin ocupar demasiada memoria.

Prompt:

Por favor, explícame la gestión de contexto en Python como si yo fuera un principiante de Python.

Resultado:

¡Claro! La gestión de contexto en Python se refiere a una técnica que nos permite automatizar la tarea de abrir y cerrar archivos o recursos que necesitamos para nuestro programa.

Imagina que tienes un archivo que quieres leer en tu programa de Python. Para hacer esto, normalmente tendrías que abrir el archivo, leer su contenido y luego cerrarlo. Sin embargo, si ocurre un error en el medio, es posible que el archivo no se cierre adecuadamente y esto podría causar problemas.

Es aquí donde entra en juego la gestión de contexto en Python. Con la gestión de contexto, podemos utilizar la palabra clave `with` para crear un contexto que se encargará de abrir y cerrar automáticamente el archivo por nosotros. El contexto se asegurará de que el archivo se cierre correctamente, incluso si se produce un error en el medio.

Aquí te muestro un ejemplo simple para que lo entiendas mejor:

```
with open('archivo.txt', 'r') as f:
    contenido = f.read()
    print(contenido)
```

En este ejemplo, utilizamos `with` para abrir el archivo 'archivo.txt' en modo lectura ('r') y lo asignamos a la variable `f`. Dentro del bloque `with`, leemos el contenido del archivo con `f.read()` y lo almacenamos en la variable `contenido`. Luego, imprimimos el contenido por pantalla.

Cuando el bloque `with` se termina, el archivo se cerrará automáticamente, sin importar si se produjo un error o no.

La gestión de contexto también puede ser utilizada para otros tipos de recursos que necesiten ser abiertos y cerrados, como conexiones a bases de datos o sockets de red.

En resumen, la gestión de contexto en Python nos permite automatizar la tarea de abrir y cerrar recursos que necesitamos en nuestro programa, lo que hace que nuestro código sea más legible y menos propenso a errores.

Creación de ficheros de entrada de prueba

A veces necesitamos abrir y leer ficheros con datos de entrada pero todavía no tenemos esos ficheros.

Podemos pedir a ChatGPT que nos cree uno con datos ficticios con solo especificarle el formato que necesitamos.

Esto nos permitiría ir desarrollando y probando el código que lee dichos datos.

Prompt:

Necesito el contenido de un fichero CSV con las siguientes columnas: nombre, apellidos, fecha de nacimiento, lugar de nacimiento, puesto de trabajo y departamento. ¿Puedes crear 10 filas con datos ficticios?

Resultado:

¡Por supuesto! Aquí te muestro el contenido del archivo CSV que necesitas:

```
nombre,apellidos,fecha de nacimiento,lugar de nacimiento,puesto de
trabajo,departamento
Luis,González,15/02/1985,Madrid,Programador,Informática
```

Sofía, Pérez, 23/04/1992, Barcelona, Ingeniero, Producción
Juan, Martínez, 05/09/1980, Valencia, Analista, Marketing
Ana, Fernández, 12/06/1995, Sevilla, Consultor, Recursos Humanos
Miguel, Rodríguez, 31/01/1988, Bilbao, Gestor, Administración
Laura, López, 18/11/1983, Málaga, Comercial, Ventas
Carlos, García, 29/07/1990, Zaragoza, Marketing, Marketing
Lucía, Sánchez, 07/03/1986, Murcia, Diseñador, Diseño
Pedro, Hernández, 22/05/1977, Granada, Recursos Humanos, Recursos Humanos
María, Gómez, 09/12/1989, Santiago de Compostela, Finanzas, Finanzas

Este archivo CSV tiene 10 filas de datos ficticios para cada columna. Puedes guardar estos datos en un archivo CSV llamado "datos.csv" y usarlos para tus necesidades.

Espero que esto te sea útil. ¡Avísame si tienes alguna pregunta!

Prompt:

¿Puedes darme los mismos datos en formato JSON? Solo tres registros esta vez.

Resultado:

¡Por supuesto! Aquí te dejo el contenido en formato JSON con tres registros:

```
[
  {
    "nombre": "Ana",
    "apellidos": "García Pérez",
    "fecha_nacimiento": "1985-06-10",
    "lugar_nacimiento": "Madrid",
    "puesto_trabajo": "Gerente",
    "departamento": "Ventas"
  },
  {
    "nombre": "Pedro",
    "apellidos": "Martínez Ruiz",
    "fecha_nacimiento": "1992-03-21",
    "lugar_nacimiento": "Barcelona",
    "puesto_trabajo": "Analista",
    "departamento": "Sistemas"
  },
  {
    "nombre": "Sara",
    "apellidos": "Gómez López",
```

```
"fecha_nacimiento": "1998-11-15",  
"lugar_nacimiento": "Valencia",  
"puesto_trabajo": "Técnico",  
"departamento": "Marketing"  
}  
]
```

Tests

Además de pedirle que nos escriba código, podemos solicitar que nos haga los tests de unidad para alguna función concreta.

En estos ejemplos pediré tests para una función que calcula el área de una circunferencia a partir de su radio.

Pediré los tests tanto en formato *unittest* como en *pytest*.

Prompt:

¿Puedes escribir tests de unidad para el siguiente código en Python?

```
import math  
  
def calcular_area_circunferencia(radio):  
    if radio < 0:  
        raise ValueError("El radio debe ser un número positivo.")  
    else:  
        area = math.pi * radio ** 2  
    return area
```

Resultado:

¡Claro! Aquí te dejo un ejemplo de tests de unidad para la función `calcular_area_circunferencia`:

```
import unittest  
import math  
  
def calcular_area_circunferencia(radio):  
    if radio < 0:  
        raise ValueError("El radio debe ser un número positivo.")  
    else:  
        area = math.pi * radio ** 2  
        return area
```

```
class TestCalcularAreaCircunferencia(unittest.TestCase):

    def test_radio_cero(self):
        self.assertEqual(calcular_area_circunferencia(0), 0)

    def test_radio_uno(self):
        self.assertEqual(calcular_area_circunferencia(1), math.pi)

    def test_radio_negativo(self):
        self.assertRaises(ValueError, calcular_area_circunferencia, -1)

    def test_radio_decimal(self):
        self.assertAlmostEqual(calcular_area_circunferencia(1.5), 7.06858347,
places=7)

if __name__ == '__main__':
    unittest.main()
```

En este caso, hemos creado una clase de tests `TestCalcularAreaCircunferencia` que hereda de `unittest.TestCase` y define cuatro métodos de test:

- `test_radio_cero`: comprueba que el área de una circunferencia con radio cero sea igual a cero.
- `test_radio_uno`: comprueba que el área de una circunferencia con radio uno sea igual a π .
- `test_radio_negativo`: comprueba que la función lance una excepción `ValueError` si el radio es un número negativo.
- `test_radio_decimal`: comprueba que la función devuelva el área correcta para un radio decimal (usando `assertAlmostEqual` para comparar números decimales con un número de decimales específico).

Por último, ejecutamos los tests con `unittest.main()` para obtener el resultado.

Prompt:

¿Puedes crear los mismos tests para la librería pytest?

Resultado:

¡Por supuesto! Aquí tienes los mismos tests de unidad, pero escritos para pytest:

```
import math
import pytest
```



```
from circunferencia import calcular_area_circunferencia

def test_radio_cero():
    assert calcular_area_circunferencia(0) == 0

def test_radio_positivo():
    assert calcular_area_circunferencia(5) == math.pi * 5 ** 2

def test_radio_negativo():
    with pytest.raises(ValueError):
        calcular_area_circunferencia(-5)
```

Asegúrate de tener la biblioteca `pytest` instalada en tu entorno de Python antes de ejecutar estos tests.

Intérprete de Python

Este es un truco maravilloso. Se trata de pedirle a ChatGPT que se comporte como si fuera un intérprete de Python.

Así podemos usarlo como si estuviéramos escribiendo código directamente en una consola de Python.

Ten en cuenta que, a pesar de que lo hace bastante bien, no es un intérprete de Python real y podría cometer algún error con código complejo.

Prompt:

Quiero que simules un intérprete de Python. Yo escribiré código Python y tu respuesta será el prompt del REPL, seguido de mi entrada y el resultado debajo. No escribas explicaciones ni ningún otro texto fuera del bloque de código. Todo lo que yo escriba será código de Python que debes ejecutar directamente en el REPL.

Respuesta:

Entendido, voy a responder con el prompt de Python seguido de la entrada y la salida correspondiente. Solo indicaré si hay algún error en el código. Por favor, escribe tu código a continuación:

Prompt:

```
a = 10
b = 20
a * b
```

Respuesta:

Aquí está el resultado de tu código:

```
>>> a = 10
>>> b = 20
>>> a * b
200
```

El código se ejecutó sin problemas y la salida correcta es 200.

Prompt:

```
def doble(a):
    return 2 * a
```

```
doble(17)
```

Respuesta:

Aquí está el resultado de tu código:

```
>>> def doble(a):
...     return 2 * a
...
>>> doble(17)
34
```

El código se ejecutó sin problemas y la salida correcta es 34.

Prompt:

```
[doble(n) for n in range(1, 10) if n % 3 == 0]
```

Respuesta:

Aquí está el resultado de tu código:

```
>>> [doble(n) for n in range(1, 10) if n % 3 == 0]
[6, 12, 18]
```

El código creó una lista que contiene el resultado de la función `doble()` para todos los números del 1 al 9 que son divisibles por 3. La salida correcta es `[6, 12, 18]`.

Consejo: podrías pedirle que no muestre tanto texto explicativo si no te interesa.

Asesor de tecnología y librerías

Si tienes algún proyectito (o projectazo) entre manos y quieres hacerte una idea inicial de qué tecnologías y librerías puedes necesitar para llevarlo a cabo, pregúntale a ChatGPT.

Prompt:

Necesito hacer desde cero una pequeña web en Python que muestre gráficas y datos históricos climatológicos. ¿Me puedes decir qué librerías, tecnologías e infraestructura podría utilizar para eso?

Resultado:

Por supuesto, existen varias opciones para crear una pequeña web que muestre gráficas y datos históricos climatológicos utilizando Python. A continuación, te daré algunas recomendaciones de librerías, tecnologías e infraestructura que podrías utilizar para este proyecto:

1. Flask: Flask es un framework de Python que te permitirá crear una aplicación web de manera rápida y sencilla. Es muy popular en la comunidad de Python para crear aplicaciones web y ofrece muchas funcionalidades para el manejo de rutas, templates, autenticación, etc.
2. Plotly: Plotly es una librería de Python que te permite crear visualizaciones interactivas en línea. Ofrece diferentes tipos de gráficas, como histogramas, barras, líneas y mapas de calor. Además, puedes personalizar las gráficas con diferentes opciones de estilo y diseño.
3. Pandas: Pandas es una librería de Python que te permite manipular y analizar datos. Es muy útil para trabajar con datos históricos climatológicos y hacer operaciones como filtrado, agregación y cálculo de estadísticas.
4. SQLite: SQLite es una base de datos ligera que podrías utilizar para almacenar los datos climatológicos. Es muy fácil de usar y no requiere una configuración complicada. Además, puedes integrar SQLite con Flask para que tu aplicación web pueda acceder a los datos almacenados en la base de datos.

5. Heroku: Heroku es una plataforma en la nube que te permite alojar tu aplicación web de manera gratuita. Es muy fácil de configurar y escalar, y ofrece muchas opciones de integración con otras tecnologías. Además, puedes usar Heroku para desplegar tu aplicación web de manera automática cada vez que hagas un cambio en el repositorio de tu proyecto.

Estas son solo algunas de las opciones que podrías utilizar para crear tu aplicación web. Espero que esta información te sea útil y te ayude a empezar con tu proyecto. ¡Buena suerte!

Y esto es todo.

Espero haberte dado algunas ideas interesantes de cómo utilizar esta increíble herramienta para que nos asista en nuestro trabajo y poder mejorar y aprender con ella.

¡Feliz programación!

Juan.-