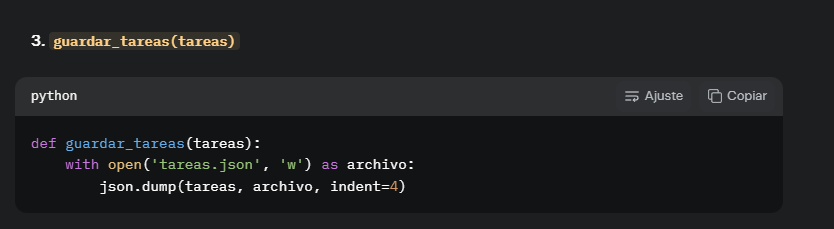


* Import json: Importa “una caja de herramientas” de Python que me permite trabajar con los archivos de tipo JSON. Los archivos JSON son una manera ordenada de guardar los datos (que en este caso serán las tareas futuras) de tal manera que el sistema pueda leerlo de manera sencilla e intuitiva
* From datetime import datetime: importamos una librería que nos permite trabajar con las fechas



Este código lo que realiza de manera principal es abrir un archivo en modo de lectura (que no se puede editar) de las tareas almacenadas. En caso de no haber archivo porque es la primera vez que se utiliza el programa, nos devolverá una lista vacía y evitamos así que el programa se haga un lío al no encontrar el archivo.

* With open(‘tareas.json’, ‘r’) as archive: Abre el archive en modo lectura. El with nos asegura que el archivo solo se cierra cuando hemos acabado con este bloque.
* Json.load(archivo): Lee el contenido del archivo. JSON y lo convierte en algo que Python entiende (como una lista de diccionarios). Como: [{“título”: “Estudiar”, “prioridad”: “alta”}], que lo carga como una lista.
* Try y except: Maneras de evitar que el programa se rompa, en caso de no haber archivo, se nos devuelve una lista vacía.



Este bloque del programa guarda las tareas una vez creadas para evitar que se pierdan al cerrar el programa. Las guarda en el archivo: tareas.json

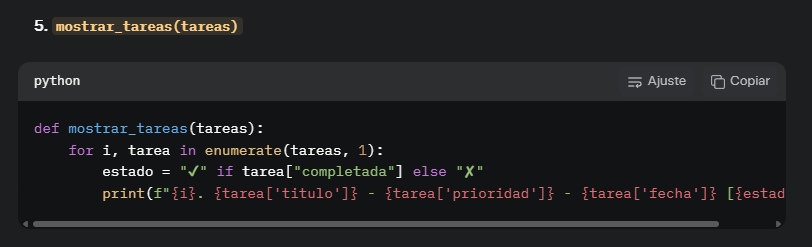
* With open(‘tareas.json’, ‘w’) as archivo: Abre el archivo en modo de escritura (que se puede editar), en caso de existir; esta función sobrescribirá el contenido anterior.
* Json.dump(tareas,archivo, indent=4):
  + Json.dump: toma todos los datos del archivo y los vuelca al archivo como texto formateado en JSON.
  + Indent=4: es una manera de estilizar los datos para que sean agradables a la vista y fáciles de interpretar por otras personas.





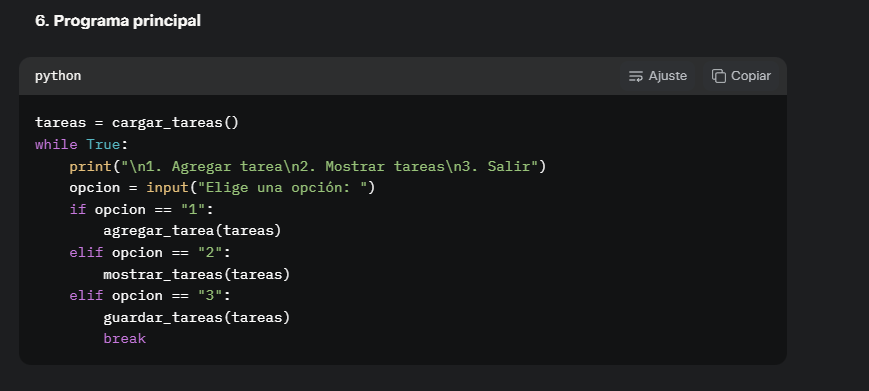
Este código lo que va a hacer es solicitar datos sobre la tarea al usuario para que las introduzca él mismo.

* Input(): solicita al usuario la información pertinente, como el título de la tarea, su fecha límite o su prioridad
* Tarea={…}: Crea el diccionario (especie de ficha de la tarea con los datos introducidos y sus valores, bien etiquetados
* Tareas.append(tarea): Añade la “ficha” a la lista de tareas



El código muestra la lista de tareas en pantalla, enumeradas y con su título, prioridad, fecha y si han sido completadas o no

* For i, tarea in enumerate(tareas, 1): Recorre la lista de tareas empezando desde el 1 (y no el 0. Es una distinción importante)
* estado = "✔" if tarea["completada"] else "✘": si la tarea se ha completado (True) muestra un tick, sino (False) muestra una X
* print: Muestra cada tarea en una línea de texto



Parte principal del programa, como su cerebro. Carga las tareas guardadas, muestra un menú y espera a que indiques que es lo que quieres hacer.

* Tareas=cargar\_tareas(tareas): Trae las tareas guardadas (o la lista vacía sino hay un archivo existente)
* While True: Repite el menú hasta que el usuario decide salir del programa
* If/elif: Condicionales; dependiendo de la elección (1,2,3 en este caso) llamará y ejecutará la función correspondiente
* Break: al elegir salir, guarda las tareas y termina el programa

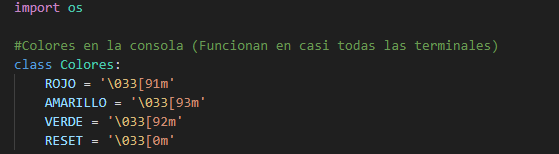
Esto sería el desglose en términos coloquiales de cómo funciona el programa en su nivel más básico. Para añadir más valor al proyecto, implementaré funcionalidades específicas para personas que poseen déficit de atención y les cuesta organizarse el tiempo, de modo que no sólo incremento la complejidad del código, sino que, al mismo tiempo me sirve a mí mismo como herramienta de trabajo. Entre estas funcionalidades se incluyen:

* Colores en consola para diferenciar las prioridades (Rojo=alta, verde=baja)
* Filtro para mostrar las tareas de hoy
* Alarma o alerta si una tarea está cerca de expirar

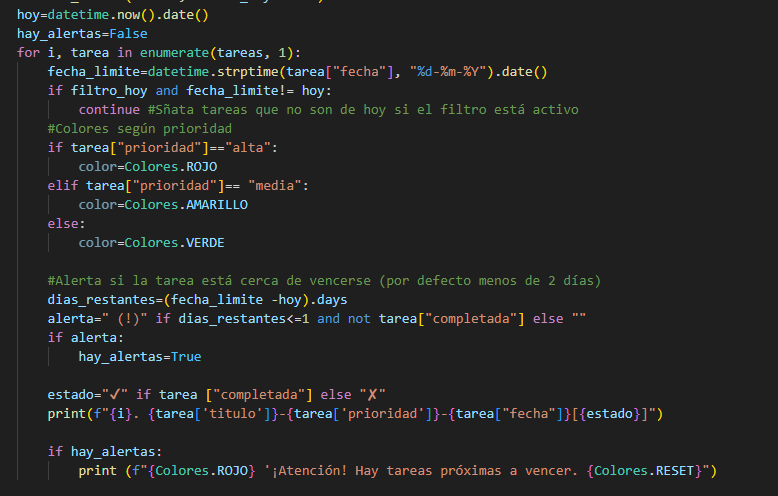
Lo primero que realizamos para implementar estas nuevas funcionalidades es importar el sistema mediante el comando de ***import os*** (que nos permite trabajar con el sistema operativo de la máquina en cuestión).

Lo segundo que hacemos es crear una clase Colores, que contiene los códigos para mostrar los textos en diferentes colores según una sencilla leyenda:

* Rojo: prioridad más alta
* Amarillo: prioridad media
* Verde: prioridad baja



Los demás cambios realizados al código se encuentran en el cuerpo principal del programa y en la función de mostrar tareas; al fin y al cabo, las funcionalidades que se están añadiendo afectan a como el usuario visualiza las alertas de las tareas registradas.



Los principales cambios realizados en esta función tienen que ver con la implementación de los colores y las alertas. Añadimos los colores y les asignamos la prioridad acorde con la leyenda establecida según el diseño de la app. El color reset hace que el color del texto vuelva al predeterminado por el sistema

En la función de mostrar\_tareas(tareas):

* Usamos los colores para asignar las prioridades y empleamos {color} al inicio del print y {Colores.reset} al final para que solo esa línea cambie de color
* Añadimos la variable del filtro\_hoy y la inicializamos en falso para que no nos muestre por defecto las tareas asignadas al día actual.
* Hoy=datetime.now().date() nos permite utilizar y pasar el día del sistema sin hacer uso de las horas
* fecha\_limite = datetime.strptime(tarea["fecha"], "%Y-%m-%d").date(): convierte la fecha de la tarea en un objeto para que sea posible compararla
* if filtro\_hoy and fecha\_limite != hoy: si el filtro está activado y la tarea no tiene como fecha límite el día de hoy, continua con la ejecución del programa (continue)
* En el programa principal añadimos una nueva opción 3 que muestre sólo las tareas de hoy

A continuación, lo que se ha realizado, es la conversión de todas las variables que se encontraban establecidas en castellano al inglés y convertir la convencionalidad de Python del snake\_case al camelCase para una legibilidad más universal, aunque la conveniencia es que en Python se emplee el snake\_case.

import json

from datetime import datetime, timedelta

import os

# Colors for console output

class Colors:

RED = '\033[91m'

YELLOW = '\033[93m'

GREEN = '\033[92m'

RESET = '\033[0m'

def loadTasks():

"""Load tasks from the JSON file."""

try:

with open('tasks.json', 'r') as file:

return json.load(file)

except FileNotFoundError:

return []

def saveTasks(tasks):

"""Save tasks to the JSON file."""

with open('tasks.json', 'w') as file:

json.dump(tasks, file, indent=4)

def addTask(tasks):

"""Add a new task to the list."""

title = input("Task title: ")

description = input("Description: ")

priority = input("Priority (high/medium/low): ").lower()

dueDate = input("Due date (DD-MM-YYYY): ")

task = {

"title": title,

"description": description,

"priority": priority,

"dueDate": dueDate,

"completed": False

}

tasks.append(task)

def showTasks(tasks, filterToday=False):

"""Display tasks with optional filter for today."""

today = datetime.now().date()

hasAlerts = False

for i, task in enumerate(tasks, 1):

# Parse due date in DD-MM-YYYY format

dueDate = datetime.strptime(task["dueDate"], "%d-%m-%Y").date()

if filterToday and dueDate != today:

continue # Skip tasks not due today if filter is active

# Set color based on priority

if task["priority"] == "high":

color = Colors.RED

elif task["priority"] == "medium":

color = Colors.YELLOW

else:

color = Colors.GREEN

# Alert if task is close to due date (within 1 day)

daysRemaining = (dueDate - today).days

alert = " (!)" if daysRemaining <= 1 and not task["completed"] else ""

if alert:

hasAlerts = True

status = "✔" if task["completed"] else "✘"

print(f"{color}{i}. {task['title']} - {task['priority']} - {task['dueDate']} [{status}]{alert}{Colors.RESET}")

if hasAlerts:

print(f"{Colors.RED}Warning! Some tasks are close to their due date.{Colors.RESET}")

# Main program

tasks = loadTasks()

while True:

os.system('cls' if os.name == 'nt' else 'clear') # Clear console

print("\n1. Add task\n2. Show all tasks\n3. Show today's tasks\n4. Exit")

choice = input("Choose an option: ")

if choice == "1":

addTask(tasks)

elif choice == "2":

showTasks(tasks)

elif choice == "3":

showTasks(tasks, filterToday=True)

elif choice == "4":

saveTasks(tasks)

break

input("\nPress Enter to continue...")

En esta parte del Código, se ha cambiado las variables a camel\_case y las variables al inglés.

A continuación, se añadió la función de validación de fechas, para comprobar que la fecha introducida es válida y no se establece fuera de lo que son los calendarios gregorianos actuales (12 meses).

Validación de fechas: Creamos la función validateDate:

* Toma una cadena (dateString) y usa datetime.strptime para intentar convertirla al formato "DD-MM-YYYY".
* Si falla (por ejemplo, "32-13-2025" o "abc"), devuelve False. Si es válida, devuelve True.
* Esto captura tanto errores de formato como fechas inválidas (como "30-02-2025", que no existe).
* **Validación en addTask**:
* Ahora, cuando pides la fecha (dueDate), usamos un bucle while True para seguir preguntando hasta que el usuario ingrese una fecha válida.
* Si la fecha no pasa la validación, mostramos un mensaje de error claro: "Invalid date format or value. Please use DD-MM-YYYY (e.g., 25-02-2025)."
* Una vez que la fecha es válida, salimos del bucle con break y creamos la tarea.
* **Feedback al usuario**:
* Añadí un print("Task added successfully!") para que sepas que la tarea se agregó bien.

Lo que se ha realizado hoy son las funcionalidades para editar y eliminar las funciones:

 **Modificación en showTasks**:

* Ahora devuelve len(tasks) (el número de tareas) para usarlo en las funciones de edición y eliminación, asegurándonos de que el usuario elija un número válido.

 **Nueva función editTask**:

* Muestra todas las tareas y pide al usuario que elija un número (restamos 1 porque las listas en Python empiezan en 0).
* Usa un bucle con try/except para validar que el número sea válido y esté en el rango.
* Permite editar cada campo (título, descripción, prioridad, fecha, estado).
* Si dejas un campo en blanco (presionas Enter), mantiene el valor actual usando el operador or.
* Valida la nueva fecha con validateDate.
* Actualiza el diccionario de la tarea en la lista.

 **Nueva función deleteTask**:

* Similar a editTask, muestra las tareas y pide un número.
* Usa tasks.pop(taskIndex) para eliminar la tarea seleccionada.
* Muestra un mensaje confirmando qué tarea se eliminó.

 **Menú actualizado**:

* Añadí las opciones "4. Edit task" y "5. Delete task", y renumeré "Exit" como 6.