# practicatddpoo

February 8, 2024

## 1 Enunciado del Ejercicio: Creación de la Clase `Dia` en Python

**Objetivo:** Desarrollar una clase `Dia` en Python que represente una fecha, la cual puede ser inicializada con valores por defecto (1 de enero de 1970) o con valores específicos de fecha (año, mes, día) proporcionados por el usuario. Esta clase debe ser implementada sin utilizar ninguna librería estándar o no estándar de Python, apoyándose únicamente en cálculos numéricos para todas sus operaciones.

**Requisitos de la Clase `Dia`:**

- **Inicialización:** La clase debe poder inicializarse tanto con valores por defecto (1 de enero de 1970) como con una fecha específica proporcionada por el usuario. Debe validar que la fecha es correcta, considerando años bisiestos.

  Es decir si creas una instancia de Día tal que asi `d = Dia()`, eso significara que d tendra los valores d.dia = 1, d.mes = 1, d.anyo = 1970

  Si intentas crear una fecha tal que `d = Dia(1970, 4, 31)` debe lanzar una excepcion de tipo ValueError con el mensaje que desees, ya que abril solo tiene 30 días

  Si intentas crear una fecha tal que `d = Dia(1970, 4, 7)` d tendrá los siguientes atributos d.dia = 7, d.mes = 4, d.anyo = 197-

- **Atributos:** La clase tendrá los atributos:
  - **dia**: dia del mes
  - **mes**: numero del mes, 1 enero, 12 diciembre
  - **anyo**: numero del año. Siempre después de cristo (desde el 1 en adelante)
  - **dia_semana**: numero del dia de la semana siendo 0 el sabado y el 6 el viernes

  representada.

- **Verificación de Fecha Correcta:** Incluir lógica para verificar la validez de la fecha, incluyendo la correcta identificación de años bisiestos.
- **Cálculo del Día de la Semana:** Implementar el algoritmo de Zeller dentro de la clase para determinar el día de la semana de la fecha.

**Restricciones:** - No se permite el uso de librerías estándar o no estándar de Python para el manejo de fechas o cualquier otro cálculo relacionado con la clase. Todos los cálculos deben realizarse de manera numérica.

### 1.1 Algoritmo de Cálculo de Años Bisiestos:

Un año es bisiesto si cumple las siguientes condiciones: - Debe ser divisible entre 4. - No será bisiesto si es divisible entre 100, a menos que también sea divisible entre 400.

Esto significa que los años divisibles entre 4 son bisiestos, con la excepción de aquellos que, siendo divisibles entre 100, no son divisibles entre 400. Por ejemplo, el año 2000 es bisiesto porque, aunque es divisible entre 100, también es divisible entre 400. En cambio, el año 1900 no es bisiesto porque, a pesar de ser divisible entre 100, no es divisible entre 400.

##Algoritmo de Zeller Detallado:

Para calcular el día de la semana, se utiliza el algoritmo de Zeller, ajustando los meses de tal manera que marzo sea el primer mes del año y febrero el último. Los pasos son los siguientes:

1. Ajustar el año y el mes: Si la fecha es en enero o febrero, resta 1 del año y considera esos meses como 13 y 14, respectivamente.
2. Calcular los siguientes valores:
   - $A = \text{año} \mod 100$
   - $B = \text{año}/100$
   - $C = 2 - B + B/4$
   - $D = A/4$
   - $E = 13 \times (\text{mes ajustado} + 1)/5$
   - $F = A + C + D + E + \text{día} - 1$
3. El resultado $F \mod 7$ te dará el día de la semana, donde 0 = sábado, 1 = domingo, 2 = lunes, etc.

**Instrucciones:**

- Implementa la clase `Dia` siguiendo los requisitos y restricciones proporcionados.
- Asegúrate de que la clase pueda validar correctamente las fechas, incluyendo la verificación de años bisiestos, y calcular el día de la semana utilizando el algoritmo de Zeller.
- Usa los test como hemos visto (opcional). Recuerda que ayudan bastante.

## 2 Enunciado del Ejercicio: Creación de la Clase `WallCalendar` en Python (opcional)

**Objetivo:** Desarrollar una clase en Python llamada `WallCalendar` que simule un calendario de pared, el cual se puede inicializar con un número específico de años, meses y días. La clase debe ajustar automáticamente los valores de meses y días para conformar una fecha válida, incluyendo la conversión de días excedentes en meses y, si es necesario, la adición de años cuando los meses excedan de 12. Esta clase también debe ser capaz de avanzar día a día y mostrar la fecha actual en formato legible.

**Requisitos de la Clase `WallCalendar`:**

1. **Inicialización:** La clase debe aceptar tres argumentos en su inicialización: `ano`, `mes`, `dia`. La lógica de inicialización debe ajustar estos valores para formar una fecha válida, teniendo en cuenta:
   - Si el número de meses es mayor a 12, se deben añadir los años necesarios.

- Los días deben ajustarse según el mes y año correspondiente, transformando cualquier exceso de días en los meses (y años si es necesario) subsiguientes.

2. **Almacenamiento de la Fecha:** Una vez inicializado, el calendario debe almacenar internamente una fecha válida basada en los cálculos anteriores.
3. **Método para Mostrar la Fecha:** Implementar un método `mostrar_fecha` que retorne la fecha actual del calendario en el formato "Día de la semana, día de mes de año" (por ejemplo, "Miércoles, 8 de abril de 1970"). Para el cálculo del día de la semana, utilizar el algoritmo de Zeller.
4. **Método para Avanzar un Día:** Incluir un método `avanza` sin argumentos que avance la fecha del calendario en un día. Este método debe ajustar automáticamente el mes y el año si es necesario.

**Restriciones:** - No se permite el uso de librerías estándar o no estándar de Python para el manejo de fechas. Todos los cálculos deben ser realizados manualmente, aplicando lógica numérica.

**Instrucciones:**

- Implementa la clase `WallCalendar` siguiendo los requisitos y restricciones detallados.
- Asegúrate de que la clase maneje correctamente la conversión de días a meses y la adición de años cuando se exceda el conteo de meses.
- La clase debe ser capaz de calcular correctamente el día de la semana usando el algoritmo de Zeller y presentar la fecha en el formato especificado.
- Incluye validación para asegurar que los números de años, meses y días sean válidos al inicializar la clase.
- Documenta tu código con comentarios que expliquen la lógica utilizada, especialmente en las partes del cálculo de la fecha válida y el ajuste de días y meses.

**Ejemplo de Uso:**

Suponiendo que se inicializa la clase `WallCalendar` con la fecha 1970, 3, 39 (representando el año 1970, 3 meses y 39 días), el calendario debe calcular y almacenar internamente la fecha correspondiente ajustada, y ser capaz de mostrarla correctamente y avanzar día a día. Es decir, al inicializarse, la fecha almacenada será el 8 de abril de 1970. Al estar en marzo de 1970 y contar 39 dias de marzo tendriamos los 31 que tiene marzo y el resto nos iriamos a abril. Por tanto es el 8 de abril.

> Date cuenta que una vez calculada la fecha correcta, puedes usar la clase Dia para obtener el dia de la semana
>
> Date cuenta tambien que a la hora de avanzar los meses debes considerar, utilizar de nuevo, el algoritmo para determinar si un año es bisiesto, y ver así si febrero tiene 28 o 29 dias.
>
> Por ejemplo `WallCalendar(1999, 13, 63)` tendriamos 13 meses, asi que nos iriamos a enero de 2000 y considerando el mes enero desde el principio, contariamos 63 dias. 31 de enero, 29 de febrero (2000) es bisiesto y 3 que sobran, la fecha seria 3 de marzo de 2000.
>
> Con la clase Dia(2000, 3, 3) obtendrias el dia de la semana