Prueba - Programación avanzada en Python

En esta prueba validaremos nuestros conocimientos de todo lo visto en el módulo. Para lograrlo, necesitarás utilizar el archivo *Apoyo Prueba - Programación avanzada en Python.zip*.

Lee todo el documento antes de comenzar el desarrollo **individual**, para asegurarte de tener el máximo de puntaje y enfocar bien los esfuerzos.

Descripción

Usted trabaja para una empresa de marketing digital que se dedica a implementar campañas publicitarias en distintas plataformas de anuncios digitales. Para ello, tienen una plataforma web para sus clientes, donde ellos ingresan los datos necesarios para la campaña que desean que sea implementada. Actualmente, esta plataforma funciona como una aplicación web MVC, construida en lenguaje PHP.

Para el año en curso, se le ha solicitado al equipo de desarrollo que construya una API que implemente la lógica "BackEnd" de su plataforma actual, de forma que ésta pueda ser consumida tanto por un FrontEnd Web de escritorio, como por su aplicación móvil, así como también por los clientes que quieran hacer uso de ella en sus propias aplicaciones. Este desarrollo se realizará de forma incremental, con entregas parciales cada dos semanas.

El equipo de desarrollo ha decidido crear esta API utilizando Python, aunque aún se está evaluando qué framework se aplicará. Para la primera etapa, se le ha asignado como primera tarea ***crear la arquitectura de clases básica que permite instanciar una campaña***, conteniendo como posibles tipos de anuncio los más solicitados por los clientes. Para ello, ***se le ha facilitado el diagrama de clases que permite codificar lo solicitado***.

Además de la información entregada por el diagrama de clases, debe considerar las siguientes reglas de negocio para implementar las operaciones y los encapsulamientos descritos en él:

De la clase Anuncio:

- Al crear, o al querer modificar el `alto` o el `ancho` de un anuncio ya creado, debe consultar si el valor que se quiere asignar es **mayor a cero**. De ser así, se asigna el valor ingresado. De no ser así, se asigna 1.
- Para esta etapa no se le solicita implementar las reglas de los atributos `url_archivo` ni `url_clic`, pero sí debe definir sus `getter` y `setter` con la lógica básica de asignación de un nuevo valor al atributo correspondiente.

- Al querer modificar el `sub_tipo` de algún anuncio ya creado, se debe validar que se esté ingresando un subtipo dentro de los permitidos en el tipo de la instancia actual. Los subtipos permitidos para las instancias de cada clase corresponden a los elementos de la tupla definida en el atributo de clase `SUB_TIPOS` respectivo. En caso de no cumplirse esta condición al momento de querer cambiar el valor del atributo `sub_tipo`, se debe lanzar una excepción `SubTipoInvalidoException`.
- El método `mostrar_formatos` es un método estático que muestra en pantalla los formatos y sus subtipos asociados disponibles para crear anuncios. Ejemplo:

  FORMATO 1:
  ==========
  - Subtipo 1
  - Subtipo 2

  Para ello debe referenciar los atributos de clase respectivos que contienen la información requerida (relación de colaboración señalada en el diagrama).

De la clase Video:

- Cada instancia de `Video` creada solo puede tener `ancho` igual a 1 y `alto` igual a 1. Estos valores no se pueden modificar.
- Al crear, o al querer modificar el atributo `duracion` de una instancia de `Video` ya creada, debe consultar si el valor que se quiere asignar es **mayor a cero**. De ser así, se asigna el valor ingresado. De no ser así, se asigna 5.
- El método `comprimir_anuncio` implementado debe mostrar por pantalla el mensaje: `"COMPRESIÓN DE VIDEO NO IMPLEMENTADA AÚN"`.
- El método `redimensionar_anuncio` implementado debe mostrar por pantalla el mensaje: `"RECORTE DE VIDEO NO IMPLEMENTADO AÚN"`.

De la clase Display:

- El método `comprimir_anuncio` implementado debe mostrar por pantalla el mensaje: `"COMPRESIÓN DE ANUNCIOS DISPLAY NO IMPLEMENTADA AÚN"`.
- El método `redimensionar_anuncio` implementado debe mostrar por pantalla el mensaje: `"REDIMENSIONAMIENTO DE ANUNCIOS DISPLAY NO IMPLEMENTADO AÚN"`.

De la clase Social:

- El método `comprimir_anuncio` implementado debe mostrar por pantalla el mensaje: `"COMPRESIÓN DE ANUNCIOS DE REDES SOCIALES NO IMPLEMENTADA AÚN"`.

- El método `redimensionar_anuncio` implementado debe mostrar por pantalla el mensaje: `"REDIMENSIONAMIENTO DE ANUNCIOS DE REDES SOCIALES NO IMPLEMENTADO AÚN"`.

De la clase Campaña:

- Se debe incluir en el constructor de la clase `Campaña` los parámetros y la lógica necesaria para instanciar ***dentro de él*** los anuncios a incorporar en el atributo que almacena el listado de anuncios. Como sugerencia, puede usar en el constructor un parámetro que sea una estructura de datos que contenga la información necesaria para crear cada instancia de `Anuncio` (por ejemplo, una tupla con diccionarios). Opcionalmente, puede refactorizar esta lógica específica en un método privado.
- Al querer modificar el `nombre` de una campaña ya creada, se debe validar que el nuevo nombre no supere los **250 caracteres**. De ser así, se debe lanzar una excepción `LargoExcedidoException`.
- Para esta etapa no se le solicita implementar las reglas de los atributos `fecha_inicio` ni `fecha_termino`, pero sí debe definir sus `setter` con la lógica básica de asignación de un nuevo valor al atributo correspondiente.
- Para esta etapa se le solicita solamente crear el método `getter` para el atributo `anuncios`, sin crear su `setter`.
- El método sobrecargado `__str__` dentro de la clase `Campaña` debe retornar una cadena de texto que informe el nombre de la campaña de la instancia actual, y la cantidad de anuncios por cada tipo. Ejemplo de retorno en una instancia específica:

```
Nombre de la campaña: Campaña 1
Anuncios: 1 Video, 2 Display, 0 Social
```

Se le solicita además crear un pequeño script *demo.py* que permita modificar los valores de los atributos de una campaña ya creada. Para ello, cree directamente en este archivo una instancia de Campaña (debe tener solo 1 anuncio de tipo Video), y luego solicite mediante `input` un nuevo `nombre` de la campaña y un nuevo `sub_tipo` para el anuncio. Envuelva ambas asignaciones en un bloque `try/except`, y en caso de que se produzca una excepción, añada su mensaje en un archivo error.log.

Requerimientos

1. **Creación de clases y atributos estáticos**
   (1 Punto)

   a. En un archivo campaña.py, definir la clase que permita crear instancias de tipo `Campaña`.
   b. En otro archivo anuncio.py, definir la clase `Anuncio` y las clases que permitan crear instancias de tipo `Video`, `Display` y `Social`, cada una de ellas con sus ***atributos de clase*** y valores correspondientes respectivos.

2. **Sobrecarga de funciones especiales**
   (2 Puntos)

   a. Sobrecargar la función especial `__init__` en clases requeridas, ***considerando toda la información entregada*** en la descripción y el diagrama de clases, de forma de poder asignar los valores recibidos por parámetro (si corresponde) en los atributos de instancia correspondientes.
   b. Sobrecargar además la función especial __str__ en la clase Campaña, considerando lo solicitado en la descripción.

3. **Implementación de especificación técnica en programación orientada a objeto**
   (3 Puntos)

   a. Aplicar ***herencia*** en clases correspondientes, incluyendo llamado a clase padre en constructor (en caso que haya sido necesario sobrescribir el constructor heredado).
   b. Definir ***métodos abstractos*** correspondientes en la clase indicada, e implementar sus sobreescrituras correspondientes en las clases indicadas en el diagrama de clases (según información entregada en la descripción).
   c. Definir ***propiedades*** con `getter` y `setter` para los atributos de instancia privados indicados en el diagrama de clases, implementando además las reglas solicitadas en la descripción.

4. **Uso de colaboración y composición**
   (2 Puntos)

   a. Aplicar ***composición*** para crear el o los componentes correspondientes, asignando al atributo correspondiente en clase(s) compuesta(s), según información entregada en diagrama de clases. *Recuerde que los componentes deben crearse dentro de la clase compuesta*. Puede usar un método privado para crear componentes.
   b. Aplicar ***colaboración,*** entre las clases adecuadas, para crear método que muestra en pantalla los formatos y subtipos disponibles para cada uno, según información entregada en la descripción y en el diagrama de clases.

5. **Flujos de excepción y escritura de archivos**
   (2 Puntos)

   a. En un archivo error.py, crear clases de ***excepciones*** solicitadas según diagrama de clases, y ***lanzarlas*** en las condiciones indicadas en la descripción.
   b. En el archivo demo.py, ***añadir en un archivo*** los mensajes de las excepciones lanzadas, según lo solicitado en la descripción.

¡Mucho éxito!