Segundo proyecto

Enunciado

En este proyecto, se espera que los estudiantes implementen un programa en Python que simule el funcionamiento de un zoológico. El programa debe hacer uso de los conceptos de programación orientada a objetos, incluyendo la creación de clases, relaciones, herencia, sobrescritura, sobrecarga, modificadores de acceso e interfaces gráficas.

El zoológico se compone de diferentes hábitat, cada una de los cuales puede contener varios animales. Cada animal tiene un nombre, una especie y un hábitat al que pertenece (entre otros).

El programa debe permitir al usuario realizar las siguientes acciones:

- Crear animales en el registro del zoológico.
- Añadir un nuevo hábitat al zoológico. Se tiene una serie definida de tipos de hábitat que se pueden crear en el zoológico: desértico, selvático, polar y acuático. Cada hábitat debe permitir conocer el número de animales asignados, temperatura, dieta y no permitir que un animal sea asignado si no cumple con las condiciones del hábitat o si no hay disponibilidad de espacio. **Crear mínimo 2 atributos específicos para cada tipo de hábitat.**
- Añadir un nuevo animal a un hábitat existente, garantizando que el hábitat exista y que permita contener el tipo de animal (cumpliendo con las condiciones del hábitat y disponibilidad).
- Listar todos los hábitat del zoológico y sus respectivos animales. Permitiendo a los usuarios ver la información de los animales, como su nombre, edad, tipo de alimentación, estado de salud y cualquier otro atributo que el estudiante haya decidido agregar.
- Realizar una acción en particular para un animal específico (dar órdenes al animal). Para esto debe tomar como parámetros el identificador del animal y el nombre de la acción a realizar. Las acciones disponibles para los animales son:
    - Comer, recibiendo el alimento y validando si este se encuentra en la lista aprobada para cada tipo de dieta.
    - Dormir, recibiendo el número de horas que el animal debe dormir y validando si es suficiente tiempo de acuerdo al valor definido para cada animal. Llevar un acumulado de sueño del animal, ya que no debería pasar el tiempo máximo de cada día.
    - Jugar, permitiendo saber si el animal ya jugó en el día o no.
- Permitir a los usuarios agregar y editar diferentes tipos de alimentos para los animales en el zoológico. Los animales deben tener diferentes tipos de alimentación según su dieta: carnívoros, herbívoros, omnívoros.
- Realizar una consulta en línea mediante un API, para mostrar información relevante para el zoológico.
- El programa debe ser capaz de manejar errores de entrada y salida de datos. Por ejemplo, si el usuario intenta ingresar una edad no válida para un animal, el programa debería informar al usuario del error y solicitar que ingrese una edad válida (informar al usuario mediante mensajes en la interfaz).
- El programa debe estar desplegado en la nube, aprovechando las funcionalidades que Streamlit tiene para facilitar este proceso.

Los estudiantes deben entregar un código Python completo que implemente todas las funcionalidades especificadas anteriormente. El proyecto se evaluará en función de la calidad del código, la implementación de los conceptos de programación orientada a objetos, la funcionalidad del programa y la documentación.

Recomendaciones

Tenga en cuenta los siguientes puntos para el desarrollo de su proyecto:

- La fecha de entrega (estimada) es el 22 de mayo de 2023 a la medianoche.
- El proyecto se puede realizar de forma individual o en parejas.

- NO SE REQUIEREN PRUEBAS UNITARIAS.
- Uso continuo de git para mantener el histórico de avance de su proyecto, con comentarios claros sobre los cambios de su programa.
- Código documentado para hacerlo claro para cualquier lector. En la documentación recuerde que sus comentarios deben explicar **por qué se hacen las cosas más que solo describir** literalmente las líneas de código.
- Buenas prácticas de programación: buen nombramiento, no números mágicos, uso de constantes.
- **Informe de autoevaluación:** Documento en el que explique qué problemas tuvo al hacer su proyecto, qué aprendió, qué le gustó, que no le gustó, qué hizo cada uno de los miembros del equipo, qué nota se pondrían de manera individual y qué nota le pondrían al compañero junto con la justificación. Cada persona del equipo debe entregar un informe individual.
- **README en el repositorio [Manual técnico]:** Presentación general del proyecto, URL del diagrama UML, explicación de cómo funciona su programa, cómo compilarlo, cómo ejecutarlo y evidencias de que cada opción funciona correctamente (imágenes). Tenga en cuenta que para documentar el README en Github debe usar MarkDown (aquí puede encontrar más info https://www.markdownguide.org).
- Todos los entregables deben ser subidos al repositorio de git del equipo. Si prefiere dar la nota de su compañero de manera anónima puede comunicarse conmigo. ***Los documentos deben tener una calidad de redacción, ortografía y presentación esperadas a nivel universitario***.

Calificación

- Autoevaluación: 10 %
- Calificación del compañero: 10%. En trabajos individuales la autoevaluación tendrá un 20% de peso. Equipos de máximo tres personas.
- Calificación compuesta por entregables, diseño, funcionalidad, estilo de codificación y mejores prácticas: 80%. La rubríca de evaluación explica los elementos que se consideran para la calificación del proyecto.
- Propiedad intelectual: valor entre 0 y 1 que multiplica la calificación total. Se demuestra durante la sustentación.
- Nota final = (criterios evaluacion%) * propiedadIntelectual

	5	**4**	**3**	**2**	**1**	**0**
Entregables (10%)	Los informes contienen toda la información solicitada y tiene alta calidad en cuanto a estilo y formato.	Se entregaron todos los informes. En términos de contenido están completos pero podría mejorar en cuanto a estilo o formato	Se entregaron todos los informes. En términos de contenido están completos pero podría mejorar en cuanto a estilo Y formato	Faltan algunos de los informes pero los entregados tiene buen estilo y formato	Faltan algunos de los informes y los entregados necesitan mejoras de estilo y formato	No se entregaron los informes
Diseño (30%)	El diseño responde a los requisitos. Se detectaron todas las clases importantes y para cada clase se detectaron los atributos y métodos importantes. Usa las relaciones correctas	El diseño responde a los requisitos. Se detectaron todas las clases importantes y para cada clase se detectaron los atributos y métodos importantes. Tiene algunas relaciones incorrectas o	El diseño responde a los requisitos. Faltó detectar algunas clases importantes. Faltó detectar algunos de los atributos y métodos importantes. Tiene algunas relaciones incorrectas. El	Faltó detectar la mayoría de clases importantes Faltó detectar muchos de los atributos y métodos importantes. Tiene muchas relaciones incorrectas	El diseño no satisface los requisitos	No se entregó el diseño

		faltantes en el diseño de los métodos como por ejemplo los atributos.	diseño no corresponde a lo codificado.			
Funcionalidad (30%)	Cumplió con todos los requisitos.	Fueron desarrollados mínimo el 75% de los requisitos	El diseño responde al 75% de los requisitos / podría mejorarse	Fueron desarrollados mínimo el 25% de los requisitos	Fueron desarrollados menos del 25% de los requisitos	La funcionalidad no responde a lo solicitado
Estilo de codificación (5%)	El código se encuentra correctamente indentado, los nombres de los atributos y las funciones cumplen con el estándar de nombramiento. El código tiene documentación interna para facilitar la revisión.	La mayoría del código se encuentra correctamente indentado- La mayoría de los nombres de los atributos y las funciones cumplen con el estándar de nombramiento. La mayoría del código tiene documentación interna para facilitar la revisión	Falta alguno de los ítems de calidad del estilo de codificación o alguna se cumple con mala calidad	Faltan dos de los ítems de calidad del estilo de codificación o dos se cumple con mala calidad	No hay código fuente suficiente para evaluar el estilo de codificación.	No cumple con el estándar de nombramiento No se encuentra correctamente indentado No está divido adecuadamente
Mejores prácticas (5%)	El código muestra mejores prácticas de desarrollo siempre. Reúso, separación de operaciones, buen manejo de ciclos, simplicidad, programación defensiva validaciones.	El código muestra mejores prácticas de desarrollo en la mayoría de los casos, pero falta mejorar algunos de los siguientes aspectos Reúso, separación de operaciones, buen manejo de ciclos, simplicidad, programación defensiva validaciones	El código muestra buenas prácticas de desarrollo pero falta mejorar dos de los siguientes aspectos Reúso, separación de operaciones, buen manejo de ciclos, simplicidad, programación defensiva validaciones	El código aplica pocas buenas prácticas de desarrollo. Falta mejorar tres de los siguientes aspectos: Reúso, separación de operaciones, buen manejo de ciclos, simplicidad, programación defensiva validaciones.	No hay código fuente suficiente para evaluar las buenas prácticas.	No se entregó proyecto o el proyecto no cubre al menos el 25% de las funcionalidades. No es posible evaluarlo.