Desafío - Estructuras de datos y funciones (III)

En este desafío validaremos nuestros conocimientos para codificar un programa en Python utilizando estilos y convenciones de programación tales como indentación, estructura de código y diccionarios. Para lograrlo, necesitarás utilizar el archivo ***Desafío evaluado - Estructuras de datos y funciones (III) (Apoyo).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

¡Llegó el gran momento! Te has unido a **ADL Desarrolladores**, una entidad dedicada a desarrollar apps entretenidas, con un equipo de desarrollo muy organizado que sigue las buenas prácticas del desarrollo de software.

Este equipo ha sido asignado para desarrollar una App en Python que permite jugar una trivia interactiva.

Esta App tendrá preguntas con 3 niveles de dificultad:

- Básica
- Intermedia
- Avanzada.

El mismo jugador define el número de preguntas a responder correspondientes a cada nivel de dificultad.

El jugador gana al responder todas las preguntas correctamente.

Las preguntas deben aparecer en un orden aleatorio, y además cada vez que alguien ejecute la app, las alternativas deben ser cambiadas de orden para evitar que alguien encuentre algún patrón de resolución.

Dado que el programa se hace bastante complejo, en una primera reunión, el project manager ha generado un backlog con tareas muy específicas, las cuales tendrán que ser desarrolladas paso a paso antes de ensamblar la app final.

Todas las subtareas consistirán en la creación de un script en Python la cual contendrá las especificaciones de una funcionalidad, y deberá ser testeada dentro del mismo archivo.

Se utilizará la siguiente estructura:

```
# Definición de Funciones de la funcionalidad

def func():
    pass

if __name__ == '__main__':
    # Test entregado en cada requerimiento (dado por el Tech Lead)
```

En el archivo `preguntas.py` se definen 3 diccionarios:

- `preguntas_basicas`
- `preguntas_intermedias`
- `preguntas_avanzadas`.

- Cada **diccionario** contiene 3 preguntas y sus correspondientes alternativas.
- Cada **pregunta** es otro diccionario que posee un enunciado (un string) y alternativas en forma de una lista anidada.
- Cada **alternativa** es una lista de dos elementos, siendo el primero el equivalente al texto de la alternativa y el segundo un indicador 0 o 1, donde 1 indica que es la alternativa correcta.

A modo de ejemplo, todas las preguntas tienen `'alt_2'` como la respuesta correcta.

Se recomienda modificar las preguntas y alternativas para simular una trivia real.

Finalmente, existe un **diccionario** llamado `pool_preguntas` que **contiene cada diccionario categorizado por nivel de dificultad**: básicas, intermedias, avanzadas.

La estructura de cada pregunta será:

```
pool_preguntas[nivel][pregunta_n]
```

Donde cada pregunta tendrá los componentes de enunciado y alternativas.

Por otra parte, el líder técnico del equipo ya desarrolló un esqueleto del funcionamiento de la App que se puede encontrar acá:
Desafío evaluado - Estructuras de datos y funciones (III) (Apoyo).zip

Este esqueleto tendrá que ir siendo rellenado dependiendo de las tareas específicas del backlog en los sectores exclusivamente detallados para ello.

 **NOTA:** Los test entregados no deben ser modificados y deben utilizarse sólo para comprobar el correcto comportamiento del programa.

Los requerimientos se detallan a continuación:

Requerimientos

1. Validador. **(1 Punto)**

Se solicita crear un programa llamado `validador.py` el cual permite validar si un valor se encuentra incluido en un conjunto de opciones.

- Se pide crear la función `validate()`, la cual debe aceptar como argumentos una **lista de opciones** y una **elección**.
- En caso de que no se ingrese una opción dentro del conjunto, la aplicación debe mostrar: `'Opción no válida, ingrese una de las opciones válidas: '` y solicitar el valor hasta que se ingrese uno válido.
- La función debe retornar la opción ingresada.

Tip: Se puede usar el operador `not in` para determinar si un elemento no es parte de una lista.

2. Escoger nivel. **(1 Punto)**

Cree un programa llamado `level.py` que incluya la función `choose_level()` que permite **escoger el nivel de dificultad** de la pregunta a realizar.

- Esta función debe aceptar como argumentos el **número de la pregunta**, y la **cantidad de preguntas por nivel** que puede ser 1, 2 o 3.
  El funcionamiento debe ser el siguiente:
  - Si se eligen 2 preguntas por nivel
    - Las preguntas n°1 y n°2 deben ser de nivel de dificultad básicas.
    - Las preguntas n°3 y n°4 de nivel intermedio
    - Y las preguntas n°5 y n°6 avanzadas.
  - En cambio si se escogen 3 preguntas por nivel la 1,2 y 3 deben ser básicas, 4,5,6 intermedias y 7, 8 y 9 avanzadas.
- La función debe retornar el nivel escogido.

3. Mezclar alternativas. **(1 Punto)**

Crear un programa llamado `shuffle.py` que contenga la función **`shuffle_alt()`**.

- Esta función debe tomar como argumento una pregunta desde el archivo `preguntas.py` (con un nivel y una pregunta definida) y mezclar las alternativas.
- La función debe retornar las alternativas mezcladas.

Tip: Recordar la función **`random.shuffle()`** de la librería random.

4. Escoger una pregunta. **(2 Puntos)**

Crear un programa llamado `question.py` que permita escoger una pregunta que no se haya hecho durante la ejecución del programa dependiendo del nivel de dificultad.

- Cree una función llamada **`choose_q()`** que tome como único argumento la dificultad de la pregunta.
- La función debe tomar las preguntas del archivo `preguntas.py` de acuerdo a la dificultad escogida.
- La función debe escoger una pregunta de las opciones disponibles y eliminar dicha opción para no volverla a escoger.
- La función debe retornar dos elementos separados, el primero debe ser el enunciado escogido y el segundo las alternativas mezcladas de acuerdo a la tarea anterior.

5. Mostrar las preguntas en pantalla. **(1 Punto)**

Crear un programa llamado `print_preguntas.py`, el cual permitirá mostrar en la app las preguntas de acuerdo a un formato:

- El programa debe contener la función **`print_pregunta()`** que tome como argumentos un enunciado y sus alternativas, y que les aplique formato.
- Esta función no debe retornar ningún objeto, sólo imprimir en pantalla.
- El formato a utilizar es imprimir el enunciado, seguido de un salto de línea.
- Luego cada alternativa irá acompañada una letra asociada, una por cada línea de la siguiente manera:
    - A. Alternativa 1
    - B. Alternativa 2
    - C. Alternativa 3
    - D. Alternativa 4

6. Verificar respuesta. **(1 Punto)**

Crear un programa llamado `verify.py` el cual debe tener la función verificar que permite comprobar si la respuesta entregada por el usuario es correcta.

- El programa debe contener la función `verificar()` que toma como argumentos las alternativas y la elección.
- En el caso que la respuesta sea correcta debe imprimir en pantalla `'Respuesta Correcta'` y retornar sólo el valor `True`, en caso contrario debe imprimir en pantalla `'Respuesta Incorrecta'` y retornar sólo el valor `False`.

7. Ensamblado de la app. **(3 Puntos)**

Como se mencionó anteriormente, el programa `main.py` incluye un esqueleto que el Tech Lead desarrolló. Este esqueleto ya incluye mensajes en el caso de acertar a una pregunta, de responder correctamente todas las preguntas o en caso de equivocarse, además de la lógica de pasar preguntas una a una.

El objetivo de esta tarea es poder incluir todas las funcionalidades desarrolladas anteriormente y completar las siguientes tareas:

1. Agregar un validador de `opcion` (el cual determina el inicio del programa o no)
2. En caso de escoger la opción 0 entonces, agregar el mensaje `'Nos vemos pronto!'` y finalizar el programa.
3. Agregar un validador al número de preguntas por nivel.
4. Escoger el nivel de la pregunta dependiendo del contador `n_pregunta` y el número de preguntas por nivel.
5. Escoger el enunciado y las alternativas dependiendo del nivel según corresponda.
6. Imprimir enunciado y sus alternativas en pantalla.
7. Validar la respuesta entregada
8. Verificar si la respuesta es correcta o no.
9. Validar los valores si desea continuar o no.

 ¡Mucho éxito!

Consideraciones y recomendaciones

- Comprime el desarrollo de cada uno de los requerimientos en un archivo .zip y luego sube tu respuesta en el LMS.