Serialización (Application checkpointing)

Nombre del alumno: José Ulises Vallejo Sierra

Código: 219747905

Sección: D06

Curso: Computación Tolerante a Fallas

Nombre del profesor: Michel Emanuel López Franco

Generar un programa que sea capaz de restaurar el estado de ejecución.

Técnica a utilizar: Módulo pickle y sus funciones dump y load para la carga y el volcado de datos

Como hemos visto a lo largo del curso, existen diversas maneras para ensanchar el código de cualquier programa o aplicación de modo que sea tolerante a los diversos fallos que se puedan presentar, el programa pasado vimos los fallos por errores de sintaxis, semánticos o de ejecución, así como sus excepciones y cómo manejarlas para hacer un código menos susceptible a fallos; en esta ocasión, la tolerancia a implementar será el uso de la serialización y deserealización que nos permiten hacer un checkpoint o un guardado de datos ingresados en alguna aplicación para posteriormente recuperarlos de algún lugar (en este caso archivo de texto binario) y cargarlos en la próxima ejecución del programa, para evitar la pérdida de información y ahorrar tiempo y esfuerzo al usuario

A continuación se presenta el programa realizado acerca de los registros de las especies de un zoológico y la captura de sus datos respectivos, así como la simulación o ejemplificación de como recuperar datos previamente ingresados después de cerrar abruptamente la ejecución por “accidente” o algún “fallo externo”, donde el programa debe ser capaz de mantener los datos a salvo para su posterior recuperación, esto haciendo uso del módulo pickle explicado más adelante.

Programa: Captura de animales en un Zoológico.

```
60  while True:
61      limpiar()
62      print("---------- MENÚ BÍTACORA ----------")
63      print("1)-Restaurar bitacora (cargar de agenda.txt)")
64      print("2)-Agregar Animal")
65      print("3)-Ver Bitacora")
66      print("0) Salir")
67      opcion = input("Eliga una opcion: ")
68
69      if opcion == "1":
70          miBitacora.retaurar_bitacora()
71          pausar()
72
73      elif opcion == "2":
74          limpiar()
75          especie = input("Especie del animal: ")
76          sexo = input("Sexo del animal: ")
77          raza = input("Raza del animal: ")
78          peso = input("Peso del animal: ")
79          tipo = input("Aéreos / Acuáticos / Terrestres: ")
80          miBitacora.añadir_animal(Animal(especie, sexo, raza, peso, tipo))
81          pausar()
82
83      elif opcion == "3":
84          limpiar()
85          miBitacora.ver_bitacora()
86          pausar()
87
88      elif opcion == "0":
89          break
```

Figura 1. Uso de un menú que ayude al usuario a la utilización de la aplicación y brinde una interfaz en consola para facilitar esto.

```
1   import pickle, os
2
3   # Funcion para limpiar la consola
4   def limpiar():
5       os.system("cls")
6
7   # Funcion para pausar la consola
8   def pausar():
9       os.system("PAUSE")
```

Figura 2. Importación de las librerías pickle (módulo para la serialización de los datos) y creación de funciones para limpiar y pausar pantalla

Creamos la clase animal, la cuál tiene todas las propiedades a especificar de éste en su constructor, y una función que ayude la visualización de los datos

```
11  class Animal:
12      def __init__(self, especie, sexo, raza, peso, tipo):
13          self.especie = especie
14          self.sexo = sexo
15          self.raza = raza
16          self.peso = peso
17          self.tipo = tipo
18          print('Se ha añadido un:',self.especie)
19
20      def __str__(self):
21          return 'Especie: {} Sexo: {} Raza: {} Peso: {} Tipo: {}'.format(self.especie, self.sexo,
22
```

Figura 3. Creación de la clase animal

Ahora, creamos la clase bitácora, la cual contará con todas las funciones para el manejo de los datos de los registros de la clase animal, primero creamos una lista de animales donde alojaremos cada animal que se irá creando, luego con la función añadir, y el parámetro donde recibirá al animal, insertamos dicho animal a la lista e invocamos el método guardar, a fin de que se guarde cada dato ingresado

```
24  class Bitacora:
25
26      animales = []
27
28      def añadir_animal(self, p):
29          self.animales.append(p)
30          self.guardar()
31
```

Figura 4. Creación de la clase bitácora y función para añadir un animal

Ahora creamos la función para restaurar la bitácora del archivo de texto binario, creamos el fichero y lo declaramos como ab+ para escribir sin sobre escribir texto binario, y posicionamos el fichero al inicio, si no se había guardado nada, imprimimos mensaje de fichero vacío, de lo contrario, hacemos el volcado del fichero a la lista con pickle.load() y por último mandamos el mensaje de cuantos animales se restauraron con la cantidad de elementos que tenga la lista.

```
33      def retaurar_bitacora(self):
34          fichero = open('bitacora', 'ab+')
35          fichero.seek(0)
36          try:
37              self.animales = pickle.load(fichero)
38          except:
39              print("El fichero está vacío")
40          finally:
41              fichero.close()
42              print("Se han cargado {} animales".format(len(self.animales)))
43
```

Figura 5. Función para restaurar los datos (volcado de datos del fichero binario)

Ahora con una función para mostrar los datos almacenados en bitácora (lista animales) mediante un for imprimimos cada elemento y esté aparecerá con el formato establecido en la clase Animal para, si la lista está vacía, se imprime dicho mensaje

```
45      def ver_bitacora(self):
46          if len(self.animales) == 0:
47              print("La bítacora está vacío")
48              return
49          for p in self.animales:
50              print(p)
```

Figura 6. Función para visualizar los datos de la bitácora

Por último, creamos la función para guardar los datos ingresados (esta se llama automáticamente después de cada animal añadido) la cual hace la carga o escritura binaria (wb) de los datos de la lista de animales al fichero con pickle.dump()

```
53      def guardar(self):
54          fichero = open('bitacora', 'wb')
55          pickle.dump(self.animales, fichero)
56          fichero.close()
```

Ejecución del programa

```
---------- MENÚ BÍTACORA -------------
1)-Restaurar bítacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bítacora
0) Salir
Eliga una opcion: 3
```

```
La bítacora está vacío
Presione una tecla para continuar . . .
```

Como observamos ingresamos la opción 3 para ver la bitácora y al no haber ingresado nada aún, nos manda el mensaje de que la bitácora está vacía

```
---------- MENÚ BÍTACORA -------------
1)-Restaurar bítacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bítacora
0) Salir
Eliga una opcion: 2
```

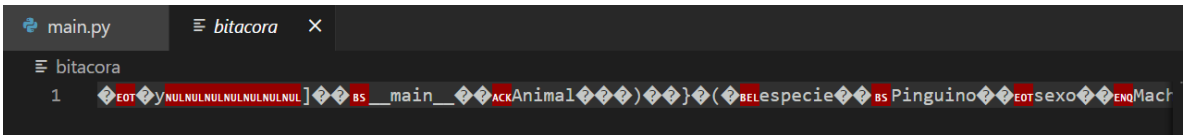
```
Especie del animal: Pinguino
Sexo del animal: Macho
Raza del animal: Emperador
Peso del animal: 5kg
Aéreos / Acuáticos / Terrestres: Terrestre
Se ha añadido un: Pinguino
Presione una tecla para continuar . . .
```

Esta vez elegimos la opción 2 para insertar un nuevo animal, nos pide sus respectivos datos y al insertarlo se nos manda el mensaje de que se ha añadido correctamente el animal.

```
---------- MENÚ BÍTACORA -------------
1)-Restaurar bítacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bítacora
0) Salir
Eliga una opcion: 3
```

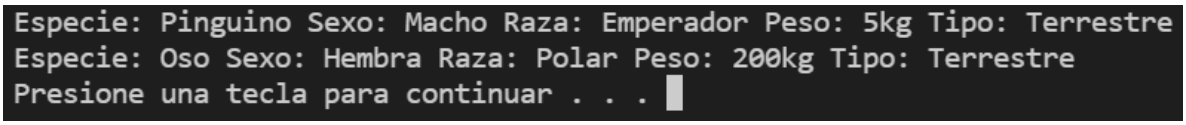
```
Especie: Pinguino Sexo: Macho Raza: Emperador Peso: 5kg Tipo: Terrestre
Presione una tecla para continuar . . .
```

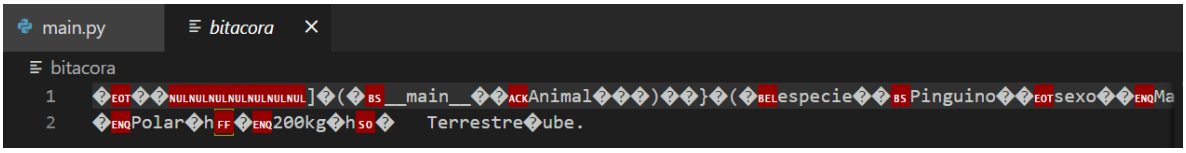
Al observar la bitácora con la opción uno del menú notamos que el animal insertado antes se encuentra agregado correctamente



También observamos que automáticamente al añadir un nuevo animal se crea un archivo de texto binario y se guardan sus datos que ingresamos, éste se va actualizando, agregando cada animal añadido.

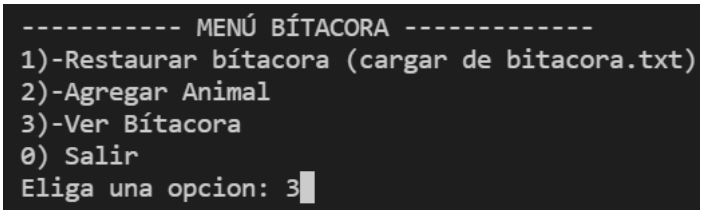
Repetimos el proceso con otro animal y notamos que se haya ingresado tanto en la bitácora como en el fichero





Ahora cerramos abruptamente la ejecución del programa desde la terminal, lo cual puede suceder accidentalmente o por un fallo externo

Volvemos a correr el programa y revisamos la bitácora



```
La bítacora está vacío
Presione una tecla para continuar . . .
```

Lógicamente está vacía ya que perdimos la información, pero presionando la opción 1 del menú, restauramos los datos que automáticamente se guardaron en el fichero, para restaurar a su vez la bitácora con los animales anteriormente ingresados.

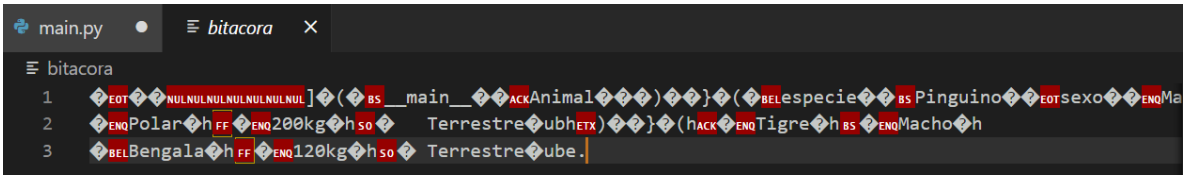
```
---------- MENÚ BÍTACORA -------------
1)-Restaurar bítacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bítacora
0) Salir
Eliga una opcion: 1
Se han cargado 2 animales
Presione una tecla para continuar . . .
```

```
Especie: Pinguino Sexo: Macho Raza: Emperador Peso: 5kg Tipo: Terrestre
Especie: Oso Sexo: Hembra Raza: Polar Peso: 200kg Tipo: Terrestre
Presione una tecla para continuar . . .
```

Y observamos que la bitácora se ha restaurado.

Ahora ingresamos un tercer animal para repetir el proceso y ver que se reestablezcan el total de los animales y no solo los nuevos añadidos en cada ejecución

```
Especie: Pinguino Sexo: Macho Raza: Emperador Peso: 5kg Tipo: Terrestre
Especie: Oso Sexo: Hembra Raza: Polar Peso: 200kg Tipo: Terrestre
Especie: Tigre Sexo: Macho Raza: Bengala Peso: 120kg Tipo: Terrestre
Presione una tecla para continuar . . .
```



Matamos la ejecución del programa y volvemos a correrlo

Y restauramos los datos desde el fichero con la opción 1, por lo que en la bitácora deberán aparecer los 3 animales ingresados hasta ahora y sus datos respectivos

```
---------- MENÚ BÍTACORA -------------
1)-Restaurar bítacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bítacora
0) Salir
Eliga una opcion: 1
Se han cargado 3 animales
Presione una tecla para continuar . . .
```

```
---------- MENÚ BÍTACORA -------------
1)-Restaurar bítacora (cargar de bitacora.txt)
2)-Agregar Animal
3)-Ver Bítacora
0) Salir
Eliga una opcion: 3
```

```
Especie: Pinguino Sexo: Macho Raza: Emperador Peso: 5kg Tipo: Terrestre
Especie: Oso Sexo: Hembra Raza: Polar Peso: 200kg Tipo: Terrestre
Especie: Tigre Sexo: Macho Raza: Bengala Peso: 120kg Tipo: Terrestre
Presione una tecla para continuar . . .
```

¡¡EXCELENTE!! Ahora nuestro programa es tolerante a la pérdida inesperada de datos gracias a la serialización y deserialización mediante el módulo pickle 😊

Conclusión

En este segundo programa, tuve que hacer un análisis profundo del tema de serialización para permitir al programa ser capaz de realizar un checkpoint o guardado de datos, para posteriormente recuperarlos desde el fichero, para ello, consulté los medios brindados por el profesor y descubrí el módulo pickle que sirve para cargar y volcar información desde el fichero, lo cual después de varios ejemplos y consultar sus métodos y funcionamiento, logré adaptar a una aplicación para el registro de especies de un zoológico.

A lo largo del desarrollo tuve varios problemas ya que pickle era un nuevo módulo para mí, pero gracias al material brindado por el docente, y la experiencia en cursos anteriores como estructura de datos, logré realizar un programa capaz de restaurar el estado de ejecución aún después de un cierre inesperado del mismo.

Sin duda una práctica que puso a reto mis habilidades a lo largo del curso y la carrera y que me servirá para seguir aprendiendo y ser base de temas futuros de la computación tolerante a fallas.

Bibliografías

- de la Vega, R. (2021, febrero 7). Introducción al módulo de Python Pickle. Pharos. https://pharos.sh/introduccion-al-modulo-de-python-pickle/
- Guzman, H. C. (s/f). Módulo pickle. Hektorprofe.net. Recuperado el 21 de febrero de 2022, de https://docs.hektorprofe.net/python/manejo-de-ficheros/modulo-pickle/
- pickle — Serialización de objetos Python — documentación de Python - 3.10.2. (s/f). Python.org. Recuperado el 21 de febrero de 2022, de https://docs.python.org/es/3/library/pickle.html

LINK AL REPOSITORIO: https://github.com/UlisesVallejo/Serializaci-n-Application-checkpointing-.git