Comenzado el	domingo, 15 de octubre de 2023, 20:22
Estado	Finalizado
Finalizado en	domingo, 15 de octubre de 2023, 20:35
Tiempo empleado	13 minutos 28 segundos
Puntos	13/14
Calificación	**9** de 10 (**91**%)

Pregunta **1**

Correcta

Se puntúa 1 sobre 1

Para cada una de las funciones/métodos para manejo archivos en Python que se indican en la columna de la izquieda, seleccione la explicación que mejor describe su uso y aplicación:

Método *tell()*	Consultar el valor del file pointer de un archivo ✔
Función *pickle.load()*	Recuperar desde un archivo serializado el contenido de una variable/objeto. ✔
Método *seek()*	Cambiar el valor del file pointer del archivo ✔
Método *flush()*	Volcar el buffer de escritura de un archivo, sin cerrarlo ✔
Función *open()*	Abrir un archivo en un modo dado entre muchos posibles ✔
Función *pickle.dump()*	Grabar el contenido de una variable/objeto en un archivo mediante serialización ✔
Método *close()*	Cerrar un archivo. De ser necesario, volcar primero el buffer de escritura en ese archivo ✔

¡Correcto!

La respuesta correcta es: Método *tell()* → Consultar el valor del file pointer de un archivo,
Función *pickle.load()* → Recuperar desde un archivo serializado el contenido de una variable/objeto.,

Método *seek()* → Cambiar el valor del file pointer del archivo,

Método *flush()* → Volcar el buffer de escritura de un archivo, sin cerrarlo,

Función *open()* → Abrir un archivo en un modo dado entre muchos posibles,

Función *pickle.dump()* → Grabar el contenido de una variable/objeto en un archivo mediante serialización,

Método *close()* → Cerrar un archivo. De ser necesario, volcar primero el buffer de escritura en ese archivo

Pregunta **2**

Correcta

Se puntúa 1 sobre 1

Para cada una de las descripciones de modos de apertura de un archivo que se indican en la columna de la izquieda, seleccione la cadena de caracteres usada en Python en la función *open()* para activar ese modo:

Archivo binario. Sólo grabación al final. El archivo se crea si no existe. Se preserva si existe. ab ✔

Archivo binario. Sólo grabación, en donde apunte el file pointer. El archivo se crea si no existe. Se pierde su contenido si existe. wb ✔

Archivo binario. Sólo lectura. El archivo debe existir. rb ✔

Archivo de texto. Sólo lectura. El archivo debe existir. rt ✔

Archivo de texto. Lectura y grabación, ambas donde apunte el file pointer. El archivo debe existir. r+t ✔

Archivo binario. Grabación al final, lectura en donde apunte el file pointer. El archivo se crea si no existe. Se preserva su contenido si existe. a+b ✔

¡Correcto!

La respuesta correcta es:
Archivo binario. Sólo grabación al final. El archivo se crea si no existe. Se preserva si existe. → ab,

Archivo binario. Sólo grabación, en donde apunte el file pointer. El archivo se crea si no existe. Se pierde su contenido si existe. → wb,

Archivo binario. Sólo lectura. El archivo debe existir. → rb,

Archivo de texto. Sólo lectura. El archivo debe existir. → rt,

Archivo de texto. Lectura y grabación, ambas donde apunte el file pointer. El archivo debe existir. → r+t,

Archivo binario. Grabación al final, lectura en donde apunte el file pointer. El archivo se crea si no existe. Se preserva su contenido si existe. → a+b

Pregunta **3**

Correcta

Se puntúa 1 sobre 1

Para cada una de las tres constantes del módulo *io* indicadas en la columna de la izquierda, seleccione su significado cuando se envían como segundo parámetro al método *seek()* de una variable *file object* que representa un archivo abierto.

io.SEEK_CUR — Suponer que el file pointer está ubicado en su posición actual ✔

io.SEEK_SET — Suponer que el file pointer está ubicado al inicio del archivo ✔

io.SEEK_END — Suponer que el file pointer está ubicado al final del archivo ✔

¡Correcto!

La respuesta correcta es:
io.SEEK_CUR → Suponer que el file pointer está ubicado en su posición actual,

io.SEEK_SET → Suponer que el file pointer está ubicado al inicio del archivo,

io.SEEK_END → Suponer que el file pointer está ubicado al final del archivo

Pregunta **4**

Parcialmente correcta

Se puntúa 0 sobre 1

Suponga que *m* es una variable tipo *file object*, que representa a un archivo ya abierto (o que será abierto cuando corresponda). ¿A qué posición (interna o externa) del archivo quedará apuntando el *file pointer* de ese archivo luego de las siguientes invocaciones al método *seek()*? (Cuando lo necesite, *suponga que el tamaño del archivo es de 20 bytes* y calcule las respuestas a partir de este dato)

`m.seek(10, io.SEEK_SET)` `m.seek(-3, io.SEEK_CUR)`	Apuntará al byte número 7. ✔
`m.seek(-5, io.SEEK_SET)`	Apuntará al byte número 25. ✖
`m.seek(5, io.SEEK_END)`	Apuntará al byte número 15. ✖
`m.seek(5, io.SEEK_SET)`	Apuntará al byte número 5. ✔
`m.seek(-5, io.SEEK_END)`	Provocará un error de runtime (file pointer negativo). ✖

Ha seleccionado correctamente 2.

La respuesta correcta es:

```
m.seek(10, io.SEEK_SET)
m.seek(-3, io.SEEK_CUR)
```

→ Apuntará al byte número 7.,

```
m.seek(-5, io.SEEK_SET)
```

→ Provocará un error de runtime (file pointer negativo).,

```
m.seek(5, io.SEEK_END)
```

→ Apuntará al byte número 25.,

```
m.seek(5, io.SEEK_SET)
```

→ Apuntará al byte número 5.,

```
m.seek(-5, io.SEEK_END)
```

→ Apuntará al byte número 15.

Pregunta **5**

Parcialmente correcta

Se puntúa 1 sobre 2

En la Ficha 25, problema número 58, se utilizó una función *buscar()* (que transcribimos más abajo) cuyo objetivo era determinar si alguno de los registros contenidos en un archivo *m* (tomado como parámetro), tenía su campo *dni* igual al valor *d* tomado también como parámetro. La función retornaba el número del byte en donde comenzaba el registro (si es que un registro con ese dni existía), o bien retornaba -1 (si el registro buscado no existía).

¿Cuáles de las siguientes afirmaciones son ciertas respecto del planteo de esta función? (Aclaración: más de una respuesta puede ser válida, por lo que marque todas las que considere correctas).

```
def buscar(m, d):
    global FD1
    t = os.path.getsize(FD1)

    fp_inicial = m.tell()
    m.seek(0, io.SEEK_SET)

    posicion = -1
    while m.tell() < t:
        fp = m.tell()
        vot = pickle.load(m)
        if vot.dni == d:
            posicion = fp
            break

    m.seek(fp_inicial, io.SEEK_SET)
    return posicion
```

Seleccione una o más de una:

- [x] a. La instrucción *fp_inicial = m.tell()* que se realiza antes del ciclo de lectura, se hace para poder recordar en qué posición estaba el file pointer del archivo m cuando *buscar()* fue invocada, de forma poder volver a posicionarlo allí con la intrucción *m.seek(fp_inicial, io.SEEK_SET)* antes de finalizar y dejar así el archivo tal como fue recibido. ✔ ¡Correcto!
- [ ] b. La función *buscar()* recibe el archivo ***m*** como parámetro pero ni lo abre ni lo cierra: el supuesto es que el archivo ***m*** viene ya abierto, y en un modo que permite lecturas. Es responsabilidad del programador que invoca a la función *buscar()* cumplir con estos supuestos, ya que de lo contrario se producirá un error de runtime y el programa se interrumpira cuando se intente leer el contenido de ***m***.
- [x] c. La instrucción *fp = m.tell()* que se realiza en el ciclo antes de leer un registro con *pickle.load()*, se hace para recordar en qué posición estaba el registro que se va a leer, de modo de poder retornarla luego si ese era el registro buscado. De otro modo, esa dirección se perdería pues *pickle.load()* mueve el file pointer al final del registro leído. ✔ ¡Correcto!
- [ ] d. La instrucción *m.seek(0, io.SEEK_SET)* ubicada antes del ciclo de lectura está allí por razones de claridad, pero no es estrictamente necesaria. Su objetivo es llevar el file pointer al inicio del archivo antes de comenzar a leerlo, pero como se supone que el archivo viene abierto en un modo que permita lecturas entonces está garantizado que el file pointer estará ya ubicado al principio.

Ha seleccionado correctamente 2.

Las respuestas correctas son:

La instrucción *fp = m.tell()* que se realiza en el ciclo antes de leer un registro con *pickle.load()*, se hace para recordar en qué posición estaba el registro que se va a leer, de modo de poder retornarla luego si ese era el registro buscado. De otro modo, esa dirección se perdería pues *pickle.load()* mueve el file pointer al final del registro leído.,

La instrucción *fp_inicial = m.tell()* que se realiza antes del ciclo de lectura, se hace para poder recordar en qué posición estaba el file pointer del archivo m cuando *buscar()* fue invocada, de forma poder volver a posicionarlo allí con la intrucción *m.seek(fp_inicial, io.SEEK_SET)* antes de finalizar y dejar así el archivo tal como fue recibido.,

La función *buscar()* recibe el archivo ***m*** como parámetro pero ni lo abre ni lo cierra: el supuesto es que el archivo ***m*** viene ya abierto, y en un modo que permite lecturas. Es responsabilidad del programador que invoca a la función *buscar()* cumplir con estos supuestos, ya que de lo contrario se producirá un error de runtime y el programa se interrumpira cuando se intente leer el contenido de ***m***.

Pregunta **6**

Correcta

Se puntúa 1 sobre 1

En la Ficha 25, problema número 59, se pidió gestionar un arreglo de registros que representaban artículos publicados en una revista científica (es decir, un arreglo con registros de tipo *Articulo*). Cada registro contenía un campo llamado *titulo* (para el título del artículo) y otro campo *codigo* (para el código numérico de identificación del artículo). El arreglo de registros debia cargarse por teclado, pero de forma que al agregar un nuevo artículo el arreglo quedase siempre ordenado de menor a mayor de acuerdo al valor del campo *titulo*. Entre las opciones a disponer en el menú, debía incluirse una para buscar un registro por su título, y otra para buscar un registro por su código numérico. La búsqueda por título se hacía con una función que aplicaba el algoritmo de búsqueda binaria, y la búsqueda por código se hacía con otra función que aplicaba búsqueda secuencial...

¿Por qué razón el programa propuesto incluyó dos funciones diferentes para las búsquedas pedidas? ¿Por qué fueron necesarios dos algoritmos diferentes para hacer las dos búsquedas requeridas en el enunciado?

Seleccione una:

a. Porque el arreglo de articulos estaba ordenado en todo momento de acuerdo al valor del campo *titulo*, y no de acuerdo al campo *codigo*. Si se puede garantizar que el arreglo estará siempre ordenado por un campo en particular (*titulo* en este caso) y permanecerá así, no hay razón para no aplicar la búsqueda binaria cuando se busque un título. Pero como el arreglo no está ordenado por el campo *codigo*, y de acuerdo al contexto del enunciado, no queda entonces remedio y debe aplicarse la búsqueda secuencial cuando se pida buscar un código. ✔ ¡Correcto!

b. En realidad, debería haberse aplicado el algoritmo de búsqueda secuencial para ambos requerimientos de búsqueda, sin importar si el arreglo está ordenado por algún campo en particular.

c. En realidad, debería haberse aplicado el algoritmo de búsqueda binaria para ambos requerimientos de búsqueda. El arreglo está ordenado, por lo que la búsqueda binaria es aplicable.

d. No hay ninguna razón para preferir un algoritmo sobre el otro, ya que ambos cumplen el objetivo pedido. El programa incluyó los dos por una razón didáctica, para mostrar al estudiante la aplicación de ambos en una misma unidad de análisis.

¡Correcto!

La respuesta correcta es:
Porque el arreglo de articulos estaba ordenado en todo momento de acuerdo al valor del campo *titulo*, y no de acuerdo al campo *codigo*. Si se puede garantizar que el arreglo estará siempre ordenado por un campo en particular (*titulo* en este caso) y permanecerá así, no hay razón para no aplicar la búsqueda binaria cuando se busque un título. Pero como el arreglo no está ordenado por el campo *codigo*, y de acuerdo al contexto del enunciado, no queda entonces remedio y debe aplicarse la búsqueda secuencial cuando se pida buscar un código.

Pregunta **7**

Correcta

Se puntúa 2 sobre 2

Suponga que el archivo *pacientes.med* ya existe y contiene varios registros del tipo registro *Paciente* grabados uno a uno por serialización en Python, mediante *pickle.dump()*. Suponga que el tipo *Paciente* es el mismo que el que se usó en la Ficha 23 con el problema número 49. Suponga ahora que se pide desarrollar un programa que use ese archivo para determinar la cantidad de días promedio que emplean entre todos los pacientes para regresar al consultorio (el promedio de los valores del campo fecha de los registros del archivo) y que una vez calculado ese promedio, se quiere mostrar por pantalla los datos de todos los pacientes que tengan el campo fecha mayor a ese promedio.

¿Es correcto el siguiente programa para cumplir con esa consigna? Si no lo es, ¿cuál es el problema?

```
import pickle
import os.path


class Paciente:
    def __init__(self, hc, nom, fec, cod):
        self.hist_clinica = hc
        self.nombre = nom
        self.fecha = fec
        self.cod_problema = cod


def display(pac):
    print('Historia clínica:', pac.hist_clinica, end=' ')
    print('Nombre:', pac.nombre, end=' ')
    print('Dias desde su última visita:', pac.fecha, end=' ')
    print('Código de enfermedad:', pac.cod_problema)


def mostrar_mayores():
    FD = 'pacientes.med'

    if not os.path.exists(FD):
        print('El archivo', FD, 'no existe...')
        print()
        return

    tbm = os.path.getsize(FD)
    m = open(FD, 'rb')

    ac, c = 0, 0
    while m.tell() < tbm:
        pac = pickle.load(m)
        ac += pac.fecha
        c += 1
    p = ac / c

    print('Pacientes con dias de re-visita mayor al promedio de días:')
    while m.tell() < tbm:
        pac = pickle.load(m)
        if pac.fecha > p:
            display(pac)

    m.close()
    print()


# script principal...
if __name__ == '__main__':
    mostrar_mayores()
```

Seleccione una:

- a. Sí. El programa es cumple correctamente la consigna.
- b. No. No es correcto: Si se van a emplear dos ciclos para leer dos veces el contenido del archivo, entonces el archivo debería haber sido abierto en modo 'r+b' y no en modo 'rb'. Así como está planteado, el segundo ciclo provocará un error de runtime y se interrumpirá el programa.

c. No. No es correcto: Cuando termina de recorrer el archivo la primera vez para calcular el promedio, el *file pointer* del archivo queda apuntando al final del mismo, por lo que el segundo ciclo no llega a activarse. No se mostrará nada en la pantalla (debería volver a 0 el file pointer antes del segundo ciclo, con *m.seek(0)*). ✔ ¡Correcto!

d. No. No es correcto: Se está calculando mal el promedio, ya que la división debería hacerse con el operador de división entera (//) en lugar del operador de división real (/). Así como está planteado, el promedio daría un resultado *float*, y no se puede comparar valores *float* con valores *int*.

¡Correcto!

La respuesta correcta es:
No. No es correcto: Cuando termina de recorrer el archivo la primera vez para calcular el promedio, el *file pointer* del archivo queda apuntando al final del mismo, por lo que el segundo ciclo no llega a activarse. No se mostrará nada en la pantalla (debería volver a 0 el file pointer antes del segundo ciclo, con *m.seek(0)*).

Pregunta **8**

Correcta

Se puntúa 2 sobre 2

¿Cuál de las siguientes instrucciones en Python insertará el valor de la variable *x* exactamente en la casilla número 4 del arreglo *v*, moviendo un casillero hacia la derecha a todos los elementos que estaban ya en *v a partir de la casilla 4 (incluida)* y sin perder a ninguno de ellos? (Suponga que el arreglo *v* tiene efectivamente al menos cuatro casilleros).

Seleccione una:

a. v[4:4] = [x] ✔ ¡Correcto!

b. v[4] += x

c. v[4] = x

d. v[4] = [x]

¡Correcto!

La respuesta correcta es:
v[4:4] = [x]

Pregunta **9**

Correcta

Se puntúa 2 sobre 2

En la Ficha 25 se presentó el problema de agregar un registro a un arreglo en Python, suponiendo que el arreglo estaba ordenado de acuerdo a un campo de los registros que contiene, y con la premisa de que la inserción se haga de forma que el arreglo se mantenga ordenado luego de la inserción. Se presentaron dos versiones para la función *add_in_order()* que llevaba a cabo esa tarea: una basada en buscar el punto de inserción mediante búsqueda secuencial, y la otra mediante búsqueda binaria. ¿Cuáles de las siguientes afirmaciones son ***ciertas*** en relación a estas soluciones? (Aclaración: más de una respuesta puede ser válida. Marque todas las que considere correctas)

Seleccione una o más de una:

- [x] a. En general, debería aplicarse la solución basada en búsqueda binaria: es claramente más rápida (veremos que su tiempo de ejecución está en función de log(n) en lugar de n) y el contexto del problema garantiza que es aplicable. ✔

  ¡Correcto! Esta es en realidad ***LA*** respuesta correcta... aunque algunas de las demás son admisibles en ciertos contextos.

- [ ] b. En general, debería aplicarse la solución basada en búsqueda secuencial: es claramente más simple desde el punto de vista de la complejidad del código fuente, y las diferencias de velocidad de ejecución respecto del algoritmo de búsqueda binaria son siempre despreciables.

- [ ] c. No debería aplicarse ninguna de las dos soluciones. Lo que debería hacerse es agregar cada nuevo registro al final del arreglo, y ordenar el arreglo luego de cada una de las inserciones.

- [x] d. La solución basada en búsqueda secuencial es más simple desde el punto de vista de la complejidad del código fuente, y podría admitirse su aplicación si se puede garantizar que el tamaño *n* del arreglo será realmente pequeño (unas pocas decenas de registros). ✔

  ¡Correcto! Aunque entienda: aceptamos que esto es admisible, pero en realidad el contexto del problema ***exige*** que se aplique búsqueda binaria... aunque sea para dejar el terreno preparado ante un eventual crecimiento inesperado del tamaño del arreglo.

¡Correcto!

Las respuestas correctas son:
En general, debería aplicarse la solución basada en búsqueda binaria: es claramente más rápida (veremos que su tiempo de ejecución está en función de log(n) en lugar de n) y el contexto del problema garantiza que es aplicable.,

La solución basada en búsqueda secuencial es más simple desde el punto de vista de la complejidad del código fuente, y podría admitirse su aplicación si se puede garantizar que el tamaño *n* del arreglo será realmente pequeño (unas pocas decenas de registros).

Pregunta **10**

Correcta

Se puntúa 1 sobre 1

En la Ficha 25 se presentó el problema de agregar un registro a un arreglo en Python, suponiendo que el arreglo estaba ordenado de acuerdo a un campo de los registros que contiene, y con la premisa de que la inserción se haga de forma que el arreglo se mantenga ordenado luego de la inserción. Se presentaron dos funciones para resolver ese problema: una buscaba el punto de inserción en forma secuencial, y la otra lo hacía con búsqueda binaria (y se dejó claro que debe ser aplicada la solución basada en búsqueda binaria).

Tanto si se usa una versión como la otra, ¿qué pasa si el arreglo ya contiene un objeto/valor igual al que se quiere insertar?

Seleccione una:

a. En ese caso, el nuevo objeto/valor será insertado en una casilla adyacente del que ya existía (si el valor que ya existía estaba en la casilla i, entonces el nuevo valor quedará en la i-1 o en la i+1 dependiendo de cómo se implemente el algorimo). ✔

¡Correcto! Y eso es claramente correcto... Si el 2 ya estaba, otro 2 quedará al lado del original...

b. En ese caso la inserción no se llevará a cabo y el arreglo permanecerá sin cambios.

c. En ese caso ambas funciones agregan el nuevo valor, pero siempre lo agregan al principio del arreglo.

d. En ese caso ambas funciones agregan el nuevo valor, pero siempre lo agregan al final del arreglo (como si se hubiese hecho directamente un append()).

¡Correcto!

La respuesta correcta es:
En ese caso, el nuevo objeto/valor será insertado en una casilla adyacente del que ya existía (si el valor que ya existía estaba en la casilla i, entonces el nuevo valor quedará en la i-1 o en la i+1 dependiendo de cómo se implemente el algorimo).

◄ Materiales Adicionales para la Ficha 25

Ir a...

Trabajo Práctico 04: Gestión de Cabinas de Peaje [4.0] ►