UAX
UNIVERSIDAD ALFONSO X EL SABIO

FUNDAMENTOS DE LA PROGRAMACIÓN I

ACTIVIDAD UNIDAD 3

Viktor Nikolov, Adrià Burgos, Yasir Doulfikar.

Índice

INTRODUCCIÓN

Para aplicar los conocimientos adquiridos durante las unidades didácticas 2 y 3, hemos elaborado un ordenador de a bordo interactivo de un vehículo que sea capaz de añadir nuevos modelos, gestionar el combustible, la autonomía y los kilómetros recorridos y mostrarlos en pantalla. Para ello, redactaremos una memoria de la practica con el fin de mostrar el proceso de creación y funcionamiento del programa.

OBJETIVOS

Poner en práctica los conocimientos adquiridos durante las unidades didácticas 2 y 3, haciendo uso de los diferentes tipos de datos y expresiones y funciones de entrada y salida de datos.

CONTENIDO DEL PROGRAMA

A continuación, mostraremos los diferentes módulos de los que se compone el programa con el fin de indicar su funcionamiento.

1. Main()
2. Coches.json()
3. showInformation()
4. addCar()
5. addOil()
6. travel()
7. resetkm()
8. deleteCar()

Main()

En primer lugar contamos con una función a la que llamamos **main(),** la cual se encarga de controlar el flujo de datos del cliente, de tal manera que muestra las diferentes opciones que tiene el cliente y gestiona la opción elegida, accediendo a los distintos módulos de los que se compone el programa, cada uno de estos módulos se ejecuta a través de un protocolo de “try/except”, de tal manera que si se produce un error en la ejecución del programa muestre por pantalla un mensaje de error notificando al cliente para que este vuelva a ejecutar el programa.

Utils()

Se trata de un conjunto de funciones las cuales nos facilitan diferentes tareas repetitivas permitiendo además gestionar los errores en los archivos del programa. En nuestro caso contamos con las siguientes funciones:

1. Doescarexists()

```python
def doesCarExist(modelName):
    data = loadJson()

    for car in data['cars']:
        if(car['modelo'] == modelName):
            return True

    return False
```

Este trozo de código DoesCarExist() simplemente nos ayuda a encontrar si nuestros coches dentro del programa existen.

2.DoesJsonexists()

```python
#Se comprueba si el JSON existe y dependiendo
def doesJsonExists():
    print('execute doesJsonExists')
    doesFileExists = os.path.exists("coches.json")
    print("coches.js existe: {}".format(str(doesFileExists)))
    if(doesFileExists == True):
        return True
    else:
        with open('coches.json', 'w') as f:
            f.write("ladsf")
```

La función del doesJsonexists() es comprobar si la base de datos (Json) existe y si se ha podido cargar correctamente, en caso contrario se encargara de crear una nueva base datos (Json) o volver a llamar el LoadJson, siempre notificando al usuario los diferentes casos de error.

3.LoadJSON()

```python
import json
#Se carga el archivo y se devulve
def loadJson():
    try:
            with open("coches.json", "r") as json_file:
                data = json.load(json_file)
                return data
    except:
            print('No se ha podido cargar correctamente la base de datos')
            return False
```

Esta función se encarga de buscar la base de datos JSON y cargar la información dentro del programa para así facilitar el uso de los datos dentro de los diferentes módulos.

(En este caso no va a ser necesario usar close ya que el statement with cierre el archivo automáticamente)

4.WriteJSON()

```
def writeJson(data):
    #Si data no es vacío se escribe su contenido en el json
    if(bool(data) == True):
        try:
            with open("coches.json", "w") as json_file:
                json.dump(data, json_file)
        except:
            print('No se han podido escribir correctamente los datos')
```

Esta función se encarga de sobrescribir en el diccionario que referencia al JSON, aquellos cambios que ha realizado el usuario dentro de los diferentes módulos, además se encarga de gestionar las excepciones en caso de error.

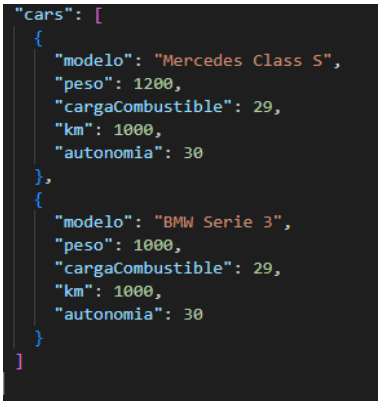
(En este caso no va a ser necesario usar close ya que el statement with cierre el archivo automáticamente)

5.ValidateInput()

```
def validateInput(inputName, inputValue):
    if((inputName == "modelName") and
        (inputValue.isdigit() == False)): return True
    if((inputName == "littersOfOil") and
        (inputValue > 0)): return True
    if((inputName == "distance") and
        (inputValue > 0)): return True
    if((inputName == "peso") and (1000>inputValue >2000)):
        return True
    return False
```

Se encarga de comprobar teniendo en cuenta dos factores, el tipo de variable en string y el valor de dicha variable, con el objetivo de revisar si dicho valor sigue las pautas correspondientes.

Coches.Json()



```
"cars": [
  {
    "modelo": "Mercedes Class S",
    "peso": 1200,
    "cargaCombustible": 29,
    "km": 1000,
    "autonomia": 30
  },
  {
    "modelo": "BMW Serie 3",
    "peso": 1000,
    "cargaCombustible": 29,
    "km": 1000,
    "autonomia": 30
  }
]
```

El JSON, es un formato estandarizado que se utiliza habitualmente para transferir datos en forma de texto que pueden enviarse a través de una red. Nuestro JSON representa objetos con nombre/valor, igual que un diccionario Python.

A través de los módulos y sus funciones podemos cargar y modificar nuestro JSON (**loadJSON** y **writeJSON).**

ShowInformation()

```
def showInformation(modelName):
    print('execution showInformation')
    data = loadJson()
    for car in data['cars']:
        if(car['modelo'] == modelName):
            print ('\nPeso:', car['peso'])
            print ('\nCombustible:', car ['cargaCombustible'])
            print ('\nKm recorridos:', car['km'])
            print ('\nAutonomia:', car['autonomia'])
            print ('')
            return True
    return False
```

El módulo **showInformation( )** debe de recibir el modelo introducido por el cliente y buscar en el JSON dicho coche **(for car in data[“cars”]),** posteriormente se encargara de acceder a los diferentes valores y los mostrará por consola en caso de encontrarlo **(if car [“modelo”] == modelName)** seguiría con el programa, sino devolvería un False y volvería al inicio mostrando un error.

Addcar()

```python
def addCar():
    print("Se ejecuta addCard")
    data = loadJson()

    """Bucle que se asegura de que el valor introducido siga los estándares definidos
    en validateInput, en caso de que alguno de estos no sea valido se resolicitará dicha
    información mostrando anteriormente un errorMessage"""
    isCarInfoValid = False
    while isCarInfoValid == False:
        carModel = input('Introduzca el nombre del modelo:  ').strip().lower()
        try:
            carWeight = float(input('Introduzca el peso del coche: '))
        except:
            print('Introduzca un número que sea válido')

        if(doesCarExist(carModel)== False):
            carInfo = {
                "modelo": carModel,
                "peso": carWeight,
                "cargaCombustible": 0,
                "km": 0,
                "autonomia": 0
            }
            data['cars'].append(carInfo)
            writeJson(data)

            print ("Su coche modelo {} dispone de 0 autonomía".format(carModel))
            print ("Su coche modelo {} ha recorrido 0 km".format(carModel))

            return True
        else:
            print("Éste modelo ya existe, introduzca uno nuevo \n")
            return False
```

El módulo **addCar( )** debe de realizar una serie de preguntas cuyas respuestas serán almacenadas y gestionas con el objetivo de crear un nuevo coche en el JSON (writeJSON)
En este caso haremos uso de un **While,** por si el cliente introduce datos no validos lo le permita avanzar y a continuación si esto se cumple, entra en un **if** donde si el cliente trata de añadir un vehículo ya existente el programa rechace dicha petición (**else**) y vuelva a solicitar nueva información, imprimiendo un mensaje de error hasta que se introduzca un modelo que aun no exista y así pueda el programa proseguir con las funciones.

Addoil()

```python
def addOil(modelName, littersOfOil):
    print('execution addOil')
    CONSUMO = 6
    data = loadJson()

    for car in data ['cars']:
        if ((car['modelo'] == modelName) and (int(car['cargaCombustible']) + littersOfOil <= 60)):
            car['cargaCombustible'] = float(littersOfOil) + car['cargaCombustible']
            car['autonomia'] = (car['cargaCombustible'])*100/CONSUMO
            autonomia = 6 * float(car['cargaCombustible'])
            print('Tiene {aut} de autonomia'.format(aut = str(autonomia)))
            writeJson(data)
            return True
        if ((car['modelo'] == modelName) and (int(car['cargaCombustible']) + littersOfOil > 60)):
            max = 60 - int(car['cargaCombustible'])
            print('Ha excedido el limite de 60 litros')
            print('Puede añadir un maximo de {maxLiters}'.format(maxLiters = str(max)))
            print('------------------------------------------------------------')
            return False
    return False
```

El módulo **addOil( )** debe de encontrar el coche que elija el cliente con la información básica del coche y añadir combustible a la variable litros hasta un máximo de 60 litros

teniendo en cuenta la variable consumo, que es una variable fija establecida en 6l a los 100 km. De nuevo haremos uso de **if** para que cuando el cliente quiera añadir combustible a su vehículo, el programa haga una serie de comprobaciones como hemos dicho antes, de que el coche exista y el combustible que ya contiene, y mostrar toda la información por pantalla a través de los **print()**

Travel()

```
def travel(modelName, distance):
    print('se ejecuta travel')
    CONSUMO = 6
    data = loadJson()

    for car in data["cars"]:
        if car["modelo"] == modelName:
            autonomia = (car["cargaCombustible"] *100 / CONSUMO)

            if distance < autonomia:
                autonomia_restante = (autonomia - distance)
                car['autonomia'] = autonomia_restante
                car["km"] = car["km"] + distance
                print("Has recorrido {} y te quedan {} de autonomía".format (distance,autonomia_restante))
                writeJson(data)
            else:
                print("No tienes suficiente combustible para recorrer {} km".format(distance))
                print('------------------------------------------------------------')

                return False
            return True
    return False
```

El módulo **travel( )** debe de comprobar antes de ejecutarse que el número de litros que dispone el coche es suficiente para el viaje que desea realizar para ello comprueba que la distancia que quiere recorrer es menor a la autonomía y si esto se cumple, se ejecutara recorriendo los km deseados modificando la autonomía, la cantidad de litros que tiene el coche y los km, añadiendo la información al JSON a través del writeJSON.

ResetKm()

```
def resetKm(modelName):
    print('execution resetKm')
    data = loadJson()
    for car in data['cars']:
        if(car['modelo'] == modelName):
            car['km'] = 0
            writeJson(data)
```

Este módulo **resetKm( )** simplemente resetea a cero dicho coche por lo que es necesario encontrar el coche e igualar la variable km a 0.

Deletecar()

```python
def deleteCar(modelName):
    print('execution deleteCar')
    data = loadJson()
    """
    Se itera en cada coche en busqueda del modelo
    que se ha de eliminar.
    Como el JSON tiene guardado a cars en un array
    he decidio crear una variable i
    la cual en cada iteración añade un +1 el
    cual nos sirve para hallar el index del objeto a
    eliminar
    """
    i = 0
    for car in data['cars']:

        if(car['modelo'] == modelName):
            del data['cars'][i]
            writeJson(data)
            return True
        i = i+1
```

A través de deletecar(), podemos borrar un coche y todos sus datos de nuestro Json. Como los coches están guardados en un array, hemos implementado en el módulo una variable “i”, que nos ayuda buscar el objeto que queremos eliminar.

AUTOEVALUACIÓN DE LA ACTIVIDAD

Una vez concluida la actividad vamos a realizar una autoevaluación con las dificultades que hemos tenido y lo que hemos podido aprender.

Por un lado, en cuanto al aprendizaje, abordar una tarea así nos ha permitido poner en practica nuestros conocimientos y chocarnos con errores de código, algo muy común en la informática, y poder encontrar soluciones y abordarlos de la mejor manera cooperando en equipo y también buscando y trabajando por cuenta propia cada integrante del grupo.

Por otro lado, las dificultades que se nos han planteado principalmente han sido errores de código dentro de nuestros módulos, los cuales hemos podido solucionar en equipo, buscando en los apuntes de la asignatura y en foros en internet.

En general, para nosotros ha sido una actividad completa y productiva para el aprendizaje y coger soltura en la materia.

BIBLIOGRAFÍA

1. Apuntes de la asignatura de Fundamentos de la Programación I.
2. https://es.stackoverflow.com/
3. *Foro de Python*. (s. f.).

   https://www.lawebdelprogramador.com/foros/Python/index1.html

4. Informáticas, J. P. (s. f.). *Foro Python – Píldoras Informáticas*.

   https://www.pildorasinformaticas.es/forums/forum/public-forum/foro-python/