Creado por:

Isabel Maniega

Librerías de Serialización

Serialización de objetos Python usando el módulo pickle

En esta sección, aprenderá a conservar objetos Python para su uso posterior.

El pickle es el proceso de preservar o extender la vida útil de los alimentos. El alimento resultante se llama encurtido y, para evitar ambigüedades, se antepone con el adjetivo 'encurtido'.

¿Alguna vez ha considerado guardar el resultado de su procesamiento de datos para su uso posterior?

La forma más sencilla de conservar los resultados es generar un archivo de texto plano y escribir los resultados. Es una forma muy sencilla de hacerlo, pero no es adecuada para conservar conjuntos de diferentes tipos de objetos o estructuras anidadas.

En Python, la serialización de objetos es el proceso de convertir una estructura de objeto en un flujo de bytes para almacenar el objeto en un archivo o base de datos, o para transmitirlo a través de una red. Este flujo de bytes contiene toda la información necesaria para reconstruir el objeto en otro script de Python.

Este proceso inverso se llama deserialización.

Los objetos de Python también pueden serializarse utilizando un módulo llamado 'pickle', y utilizando este módulo, puede 'encapsular' sus objetos de Python para su uso posterior.

El módulo 'pickle' es un módulo muy popular y conveniente para la serialización de datos en el mundo de los Pythonistas.

Entonces, ¿qué se puede encapsular y luego desencapsular?

Se pueden encapsular los siguientes tipos:

- Ninguno, booleanos;
- enteros, números de punto flotante, números complejos;
- cadenas, bytes, bytearrays;
- tuplas, listas, conjuntos y diccionarios que contienen objetos encapsulables;
- objetos, incluidos objetos con referencias a otros objetos (¡recuerde evitar los ciclos!)
- referencias a funciones y clases, pero no a sus definiciones.

Encapsularemos nuestro primer conjunto de datos que consta de:

- un diccionario anidado que contiene información sobre monedas;
- una lista que contiene una cadena, un entero y una lista.

Cuando ejecute el código presentado en el panel derecho, se debe crear un nuevo archivo. Recuerde ejecutar el código localmente.

El código comienza con la declaración de importación responsable de cargar el módulo pickle:

```
import pickle
```

Más adelante, podrá ver que el identificador de archivo 'file_out' está asociado con el archivo abierto para escritura en modo binario. Es importante abrir el archivo en modo binario, ya que estamos volcando datos como un flujo de bytes:

```
with open('multidata.pckl', 'wb') as file_out:
```

Ahora es el momento de conservar el primer objeto con la función `dump()`. Esta función espera que se conserve un objeto y un identificador de archivo.

```
pickle.dump(a_dict, file_out)
```

Y el segundo objeto se conserva de la misma manera:

```
pickle.dump(a_list, file_out)
```

En este resultado, hemos creado un archivo que conserva los objetos pickle.

In [1]:
```
import pickle

a_dict = dict()
a_dict['EUR'] = {'code':'Euro', 'symbol': '€'}
a_dict['GBP'] = {'code':'Pounds sterling', 'symbol': '£'}
a_dict['USD'] = {'code':'US dollar', 'symbol': '$'}
a_dict['JPY'] = {'code':'Japanese yen', 'symbol': '¥'}

a_list = ['a', 123, [10, 100, 1000]]

with open('multidata.pckl', 'wb') as file_out:
    pickle.dump(a_dict, file_out)
    pickle.dump(a_list, file_out)
```

Ahora es el momento de descomprimir el contenido del archivo.

El código presentado es bastante simple:

- estamos importando un módulo pickle;
- el archivo se abre en modo binario y el identificador de archivo se asocia con el archivo;

- leemos consecutivamente algunas porciones de datos y las deserializamos con la función `load()`;
- finalmente, examinamos el tipo y el contenido de los objetos.

Presta atención al hecho de que con el módulo 'pickle', debes recordar el orden en el que se conservaron los objetos y el código de deserialización debe seguir el mismo orden.

In [2]:
```
import pickle

with open('multidata.pckl', 'rb') as file_in:
    data1 = pickle.load(file_in)
    data2 = pickle.load(file_in)

print(type(data1))
print(data1)
print(type(data2))
print(data2)
```

```
<class 'dict'>
{'EUR': {'code': 'Euro', 'symbol': '€'}, 'GBP': {'code': 'Pounds sterlin
g', 'symbol': '£'}, 'USD': {'code': 'US dollar', 'symbol': '$'}, 'JPY':
{'code': 'Japanese yen', 'symbol': '¥'}}
<class 'list'>
['a', 123, [10, 100, 1000]]
```

Al principio del módulo de serialización, mencionamos que los objetos serializados se pueden conservar en una base de datos o enviarse a través de una red. Esto implica otras dos funciones correspondientes a las funciones pickle.dumps() y pickle.loads():

- `pickle.dumps(object_to_be_pickled)` – espera un objeto inicial, devuelve un objeto byte. Este objeto byte debe pasarse a una base de datos o controlador de red para conservar los datos;
- `pickle.loads(bytes_object)` – espera el objeto bytes, devuelve el objeto inicial.

En el panel derecho se presenta un ejemplo de serialización y deserialización in situ.

In [3]:
```
import pickle

a_list = ['a', 123, [10, 100, 1000]]
bytes = pickle.dumps(a_list)
print('Intermediate object type, used to preserve data:', type(bytes))

# now pass 'bytes' to appropriate driver

# therefore when you receive a bytes object from an appropriate driver yo
b_list = pickle.loads(bytes)
print('A type of deserialized object:', type(b_list))
print('Contents:', b_list)
```

```
Intermediate object type, used to preserve data: <class 'bytes'>
A type of deserialized object: <class 'list'>
Contents: ['a', 123, [10, 100, 1000]]
```

Recuerde que los intentos de picklear objetos que no se pueden picklear generarán la excepción `PicklingError`.

Intentar picklear una estructura de datos altamente recursiva (tenga en cuenta los ciclos) puede exceder la profundidad de recursión máxima, y se generará una excepción `RecursionError` en tales casos.

Tenga en cuenta que las funciones (tanto las integradas como las definidas por el usuario) se pickean por su referencia de nombre, no por ningún valor. Esto significa que solo se pickea el nombre de la función; no se pickea ni el código de la función ni ninguno de sus atributos de función.

De manera similar, las clases se pickean por referencia nombrada, por lo que se aplican las mismas restricciones en el entorno de despicking. Tenga en cuenta que no se pickea ningún código ni dato de la clase.

Esto se hace a propósito, para que pueda corregir errores en una clase o agregar métodos a la clase, y aún así cargar objetos que se crearon con una versión anterior de la clase.

Por lo tanto, su función es garantizar que el entorno en el que se descomprime la clase o función pueda importar la definición de la clase o función. En otras palabras, la función o clase debe estar disponible en el espacio de nombres de su código que lee el archivo de pickle.

De lo contrario, se generará una excepción `AtrributeError`.

El siguiente código demuestra la situación del pickle de la definición de función:

```
import pickle

def f1():
    print('Hello from the jar!')

with open('function.pckl', 'wb') as file_out:
    pickle.dump(f1, file_out)
```

No vemos ningún error, por lo que podemos concluir que f1() se ha procesado correctamente y ahora podemos recuperarlo del archivo.

Ejecuta el código en el editor y observa qué sucede.

In [4]:
```
import pickle

def f1():
    print('Hello from the jar!')

with open('function.pckl', 'wb') as file_out:
    pickle.dump(f1, file_out)
```

In [5]:
```
import pickle
```

```
with open('function.pckl', 'rb') as file_in:
    data = pickle.load(file_in)

print(type(data))
print(data)
data()
```

```
<class 'function'>
<function f1 at 0x76f63ae3a9e0>
Hello from the jar!
```

A continuación se muestra el mismo ejemplo sobre la definición de clase y el picking de objetos:

```
import pickle

class Cucumber:
    def __init__(self):
        self.size = 'small'

    def get_size(self):
        return self.size

cucu = Cucumber()

with open('cucumber.pckl', 'wb') as file_out:
    pickle.dump(cucu, file_out)
```

No vemos ningún error, por lo que podemos concluir que la clase y el objeto Cucumber se procesaron correctamente y ahora podemos recuperarlos del archivo. De hecho, solo se conserva el objeto, pero no su definición, lo que nos permite determinar la disposición de los atributos:

Si ejecuta el código, recibirá:

In [6]:
```
import pickle

class Cucumber:
    def __init__(self):
        self.size = 'small'

    def get_size(self):
        return self.size

cucu = Cucumber()

with open('cucumber.pckl', 'wb') as file_out:
    pickle.dump(cucu, file_out)
```

In [7]:
```
import pickle

with open('cucumber.pckl', 'rb') as file_in:
    data = pickle.load(file_in)

print(type(data))
print(data)
```

```
print(data.size)
print(data.get_size())
```

```
<class '__main__.Cucumber'>
<__main__.Cucumber object at 0x76f63ac5b760>
small
small
```

Resumen

Algunas palabras adicionales sobre el módulo pickle:

- es una implementación de Python del proceso de serialización, por lo que los datos enlatados no se pueden intercambiar con programas escritos en otros lenguajes como Java o C++. En tales casos, debe pensar en los formatos JSON o XML, que pueden ser menos convenientes que el pickling, pero cuando se asimilan son más poderosos que el pickling;
- el módulo pickle evoluciona constantemente, por lo que el formato binario puede diferir entre las diferentes versiones de Python. Preste atención a que tanto las partes que serializan como las que deserializan están haciendo uso de las mismas versiones de pickle;
- el módulo pickle no está protegido contra datos erróneos o construidos maliciosamente. Nunca desempaquete los datos recibidos de una fuente no confiable o no autenticada.

Serialización de objetos Python usando el módulo shelve

Como recordará, el módulo pickle se usa para serializar objetos como un flujo de un solo byte. Tanto las partes que serializan como las que deserializan deben respetar el orden de todos los elementos colocados en un archivo o base de datos, o enviados a través de una red.

Hay otro módulo útil, llamado **shelve**, que está *construido sobre pickle* e implementa un diccionario de serialización donde los objetos se seleccionan y se asocian con una clave. Las claves deben ser cadenas comunes, porque la base de datos subyacente (dbm) requiere cadenas.

Por lo tanto, puede abrir su archivo de datos almacenado y acceder a sus objetos seleccionados a través de las claves de la misma manera que lo hace cuando accede a los diccionarios de Python. Esto podría ser más conveniente para usted cuando está serializando muchos objetos.

Usar shelve es bastante fácil e intuitivo.

Primero, importemos el módulo apropiado y creemos un objeto que represente una base de datos basada en archivos:

```
import shelve
my_shelve = shelve.open('first_shelve.shlv', flag='w')
```

El significado del parámetro opcional flag:

Valor	Significado
'r'	Abrir base de datos existente solo para lectura
'w'	Abrir base de datos existente para lectura y escritura
'c'	Abrir base de datos para lectura y escritura, creándola si no existe (este es un valor predeterminado)
'n'	Siempre crear una base de datos nueva, vacía, abierta para lectura y escritura

Ahora nuestro objeto shelve está listo para la acción, así que insertemos algunos elementos y cerremos el objeto shelve.

```
my_shelve['USD'] = {'code':'US dollar', 'symbol': '$'}
my_shelve['JPY'] = {'code':'Japanese yen', 'symbol': '¥'}
my_shelve.close()
```

Ahora abramos el archivo shelve para demostrar el acceso directo a los elementos (contrario al acceso secuencial a los elementos cuando usamos pickles).

```
new_shelve = shelve.open(shelve_name)
print(new_shelve['USD'])
new_shelve.close()
```

El código final se presenta en el panel derecho.

In [8]:
```
import shelve

shelve_name = 'first_shelve.shlv'

my_shelve = shelve.open(shelve_name, flag='c')
my_shelve['EUR'] = {'code':'Euro', 'symbol': '€'}
my_shelve['GBP'] = {'code':'Pounds sterling', 'symbol': '£'}
my_shelve['USD'] = {'code':'US dollar', 'symbol': '$'}
my_shelve['JPY'] = {'code':'Japanese yen', 'symbol': '¥'}
my_shelve.close()

new_shelve = shelve.open(shelve_name)
print(new_shelve['USD'])
new_shelve.close()
```

```
{'code': 'US dollar', 'symbol': '$'}
```

Debes tratar un objeto shelve como un diccionario Python, con algunas notas adicionales:

- las claves deben ser cadenas;
- Python coloca los cambios en un búfer que se vacía periódicamente en el disco. Para hacer que se vacíe de forma inmediata, llama al método sync() en tu objeto shelve;
- cuando llamas al método close() en un objeto shelve, también vacía los búferes.

Cuando tratas un objeto shelve como un diccionario Python, puedes hacer uso de las utilidades del diccionario:

- la función len();
- el operador in;
- los métodos keys() e items();
- la operación update, que funciona igual que cuando se aplica a un diccionario Python;
- la instrucción del, que se utiliza para eliminar un par clave-valor.

Después de ejecutar el código, notarás además que se crean algunos archivos para respaldar la base de datos. No intentes alterar esos archivos con utilidades externas, porque tu shelve puede volverse inconsistente, lo que resultará en errores de lectura/escritura.

El uso de shelve es realmente fácil y efectivo. Además, debes saber que puedes simular el shelve mediante el pickle de todo el diccionario, pero el módulo shelve utiliza la memoria de manera más eficiente, por lo que siempre que necesites acceder a objetos pickle, utiliza un shelve.

Y la observación final es:

- debido a que el módulo shelve está respaldado por pickle, no es seguro cargar un shelve desde una fuente no confiable. Al igual que con pickle, cargar un shelve puede ejecutar código arbitrario.

Serialización de objetos Python usando el módulo marshal

El módulo Marshal es utilizado por Python para la serialización interna de objetos de Python, con el fin de admitir operaciones de lectura y escritura en versiones compiladas de módulos de Python (archivos .pyc). Las características de serialización de objetos del módulo Marshal son las mismas que las del módulo pickle.

El formato de datos que utiliza el módulo Marshal no es compatible con las versiones de Python. Por lo tanto, un script de Python compilado (archivo .pyc) de una versión probablemente no funcione para otra.

Para leer y escribir objetos serializados desde/hacia archivos, el módulo marshal define las funciones load() y dump().

Función dump()

Esta función se utiliza para escribir una representación de bytes de un objeto Python en un archivo con permiso de escritura.

Sintaxis:

```
marshal.dump(valor, archivo[, versión])
```

Dónde:

- El valor debe ser un tipo compatible.
- El archivo debe ser un archivo binario escribible .
- El argumento de versión indica el formato de los datos.

Si el valor tiene un tipo no admitido, se genera una excepción `ValueError`.

Función load()

Esta función lee un valor (datos de byte) de un archivo binario abierto y devuelve el objeto Python convertido.

Sintaxis:

```
mariscal.load(archivo)
```

Dónde,

- El archivo debe ser un archivo binario legible .

Si no se lee ningún valor, se genera `EOFError`, `ValueError` o `TypeError`.

Función compile()

El código utiliza la función compile() incorporada que devuelve el objeto de código Python desde la fuente.

Sintaxis:

```
compile(source, filename, mode, flag, dont_inherit,
optimize)
```

Dónde,

- source -. puede ser un objeto String , un objeto Bytes o un objeto AST .
- filename — El nombre del archivo
- mode — exec or eval or single.
- flag(opcional) — Cómo compilar el objeto fuente. Predeterminado 0.
- dont_inherit (opcional): cómo compilar el objeto de origen. Valor predeterminado: Falso.
- optimize (opcional): nivel de optimización del compilador. Predeterminado: -1.

El parámetro de mode es:

- 'exec ' si la fuente contiene una secuencia de declaraciones,
- 'eval' si hay una sola expresión
- 'single' si contiene una sola declaración interactiva.

El objeto de código de compilación se almacena en un archivo .pyc utilizando la función dump() .

El siguiente programa muestra el uso de las funciones dump() y load().

In [9]:
```
import marshal

script = """
a = 10
b = 20
print('addition=', a + b)
"""
code = compile(script, "script", "exec")
f = open("dev.pyc","wb")
marshal.dump(code, f)
f.close()
```

La función `load()` se utiliza para deserializar el objeto del archivo .pyc. Dado que devuelve un objeto de código Python, se puede ejecutar mediante la función `exec()` u otra función incorporada.

In [10]:
```
import marshal

f=open("dev.pyc","rb")
data=marshal.load(f)
exec(data)
```

```
addition= 30
```

Serialización de objetos Python usando el módulo Cpickle

La serialización es un proceso de almacenamiento de un objeto como un flujo de bytes o caracteres para transmitirlo a través de una red o almacenarlo en el disco para recrearlo junto con su estado cuando sea necesario. El proceso inverso se denomina deserialización.

En Python, el módulo Pickle nos proporciona los medios para serializar y deserializar los objetos de Python. Pickle es una biblioteca potente que puede serializar muchos objetos complejos y personalizados que otras bibliotecas no pueden hacer. Al igual que Pickle, existe un módulo cPickle que comparte los mismos métodos que Pickle, pero está escrito en C. El módulo cPickle está escrito como una función de C en lugar de un formato de clase.

Nota: cPickle era una biblioteca de Python que proporcionaba una implementación más rápida de la biblioteca pickle, que se utiliza para serializar y deserializar objetos de Python.
cPickle ha quedado obsoleto en favor de la biblioteca pickle, que ahora está implementada en C y ofrece beneficios de rendimiento similares.

Diferencia entre Pickle y cPickle:

- Pickle utiliza una implementación basada en clases de Python, mientras que cPickle está escrito como funciones de C. Como resultado, cPickle es mucho más rápido que pickle.

- Pickle está disponible tanto en Python 2.x como en Python 3.x, mientras que cPickle está disponible en Python 2.x de forma predeterminada. Para usar cPickle en Python 3.x, podemos importar _pickle.
- cPickle no admite subclases de pickle. cPickle es mejor si la subclasificación no es importante; de lo contrario, Pickle es la mejor opción.

Dado que tanto pickle como cPickle comparten la misma interfaz, podemos utilizar ambos de la misma manera. A continuación se muestra un código de ejemplo como referencia:

In [11]:
```
# importar: _pickle
import _pickle as pckl

data = [{ 'a':'A', 'b':2, 'c':3.0 }]
print('DATA:', data)

# Serialización como objeto (dumps):

data_string = pckl.dumps(data)
print('PICKLE text:', data_string)
```

```
DATA: [{'a': 'A', 'b': 2, 'c': 3.0}]
PICKLE text: b'\x80\x04\x95#\x00\x00\x00\x00\x00\x00\x00]\x94}\x94(\x8c\x0
1a\x94\x8c\x01A\x94\x8c\x01b\x94K\x02\x8c\x01c\x94G@\x08\x00\x00\x00\x00\x
00\x00ua.'
```

In [12]:
```
# Deserialización como objeto (loads):

readata = pckl.loads(data_string)
readata
```

Out[12]:
```
[{'a': 'A', 'b': 2, 'c': 3.0}]
```

In [13]:
```
# Serialización como archivo (dump):

file = open('datafile.txt', 'wb')
pckl.dump(data, file)
file.close()
```

In [14]:
```
# Deserialización como archivo (load):

file = open('datafile.txt', 'rb')
info = pckl.load(file)
print(info)
file.close()
```

```
[{'a': 'A', 'b': 2, 'c': 3.0}]
```

Creado por:

Isabel Maniega