

Python: File Handling [Binary Files]

What is Binary file?

In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language. Operations on binary files are faster as no translation required. Image files such as .jpg, .png, .gif, etc., and documents such as .doc, .xls, .pdf, etc., all of them constitute binary files. Some of the most common access modes are listed below:

Some of the most common access modes are listed below:

Binary File Mode	Use	Description
rb	Read only	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
Wb	Write only	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
ab	Append	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
rb+ or r+b	Read and write	Opens a file for both reading and writing in binary format. In this mode file must exist otherwise error is raised. The file pointer placed at the beginning of the file.
w+b or wb+	Write and read	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
ab+ or a+b	Write and read	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
wb+ or		Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Python: pickle module :

Python pickle module is used for *serializing and de-serializing* a *Python object structure*. Any object in *Python* can be *pickled* so that it can be *saved on disk*. *Pickling* is a way to *convert* a *Python object (list, dict, etc.)* into a *character stream*.

The pickle provides us with the ability to serialize (by using dump () method) and desterialize (by using load () method) objects, i.e., to convert objects into bit streams which can be stored into files and later be used to reconstruct the original objects.

Note:

The *pickle module* implements a fundamental, but powerful algorithm for *serializing and de-serializing a Python object structure*. “*pickling*” is the process whereby a *Python object hierarchy is converted into a byte-stream*, and “*unpickling*” is the inverse operation, whereby a *byte-stream is converted back into an object hierarchy*

There are some *data types* which *pickle* cannot *serialize*, but it is still capable of *serializing most of the objects typically used in Python programs*. A comprehensive list of data types which pickle can serialize:

- None, True, and False.
- Integers, floating point numbers, complex numbers.
- Strings, bytes, bytearrays.
- Tuples, lists, sets, and dictionaries containing only picklable objects.
- Functions defined at the top level of a module (using def, not lambda).
- Built-in functions defined at the top level of a module.
- Classes that are defined at the top level of a module.

In order to work with *pickle module*, we must have to *import pickle* in the program.

Writing instances / objects into a File – Pickling:

In order to write an *object* on *to a file*, opened in *write mode*, we can use the *dump()* function as per following syntax:

```
Pickle.dump(<object-to-be-written>, <filehandle -of-open-file>)  
pickle.dump(object_to_be_written, fileObject)
```

pickle.dump() function to store the *object* data to the file.

pickle.dump () method is used to write object in the file or to *serialize* an *object hierarchy*.

Writing Instance onto a file [Using pickle.dump()]:

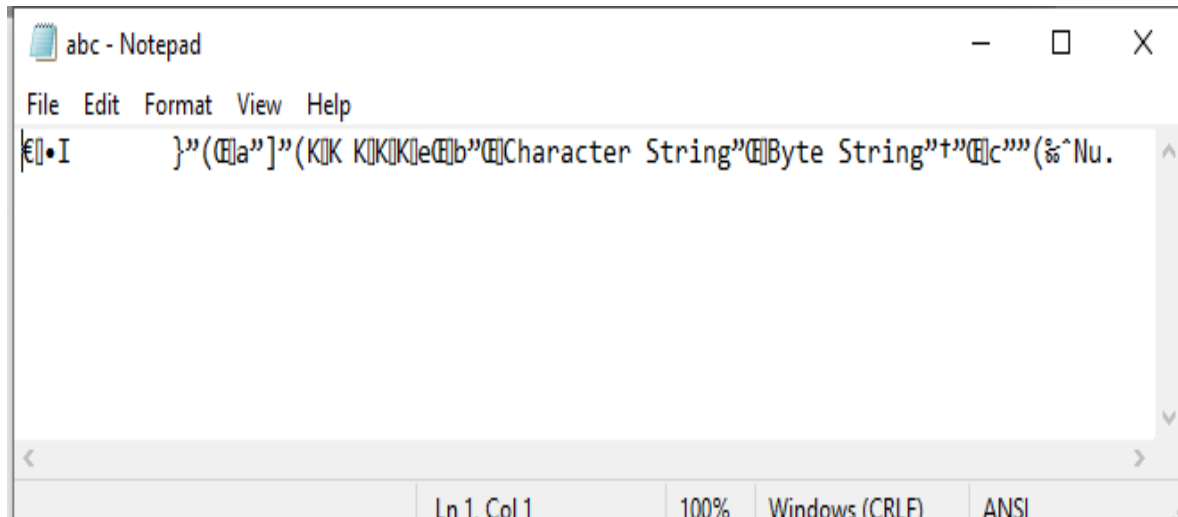
Example: Program to write structure/dictionary to the binary file.

```
import pickle  
data = {'a': [1, 2, 3, 4, 9+12],  
        'b': ("Character String", "Byte String"),  
        'c': {None, False, True}  
        }
```

```
FILE= open('abc.dat','wb')  
pickle.dump(data,FILE)  
FILE.close()
```

```
# OR  
"""  
with open('abc.dat','wb') as FILE:  
    pickle.dump(data, FILE)  
"""
```

The contents of the generated binary file are encrypted (serialized) in binary as:



Reading Instance from a binary file Using pickle.load():

In order to *retrieve pickled data*, we have to use *pickle.load()* function. The *pickle.load()* method *unpickle* or *unserialize* the data coming from the *file*.

Syntax of load():

Object = load (fileObject)

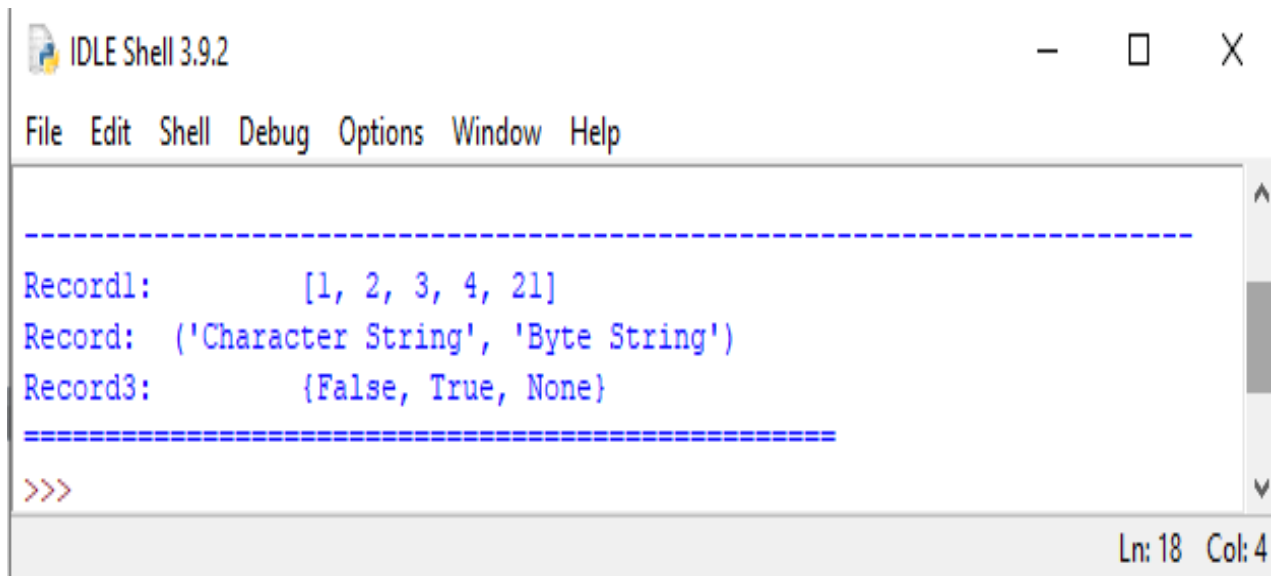
The primary argument of pickle load function is the file object that we get by opening the file in read-binary (rb) mode.

Example: To read dictionary from the binary file “abc.dat”

```
import pickle
F=open("abc.dat",'rb')
data = pickle.load(F)
print('\n-----')
print('Record1:\t',data['a'], '\nRecord:\t',data['b'],'\nRecord3:\t', data['c'])
print('=====')
F.close()

# OR
'''
with open('abc.dat','rb') as FILE:
    data = pickle.load(FILE)
    print('-----')
    print('Record1:\t',data['a'], '\nRecord2:\t',data['b'],'\nRecord3:\t', data['c'])
    print('=====')
'''
```

Output:



The screenshot shows a window titled "IDLE Shell 3.9.2" with a standard menu bar (File, Edit, Shell, Debug, Options, Window, Help). The main text area contains the following output in blue text:

```
-----  
Record1:      [1, 2, 3, 4, 21]  
Record:  ('Character String', 'Byte String')  
Record3:      {False, True, None}  
=====
```

Below the output is a red prompt ">>>" on a new line. The status bar at the bottom right indicates "Ln: 18 Col: 4".