Operator Functions Set 1 & Set 2
Data Types
Introduction
Arrays Set 1, Set 2
String Methods Set 1, Set 2, Set 3
String Template Class & String Formatting using %
List Methods Set 1, Set 2, Set 3
Tuples & Sets
Dictionary Methods Set 1, Set 2
ChainMap
Explore More...
Control Flow
Loops and Control Statements
Counters & Accessing Counters
Iterators & Iterator Functions Set 1, Set 2
Generators
Explore More...
Functions
Function Decorators
Returning Multiple Values
Yield instead of Return
Python Closures & Coroutine
Explore More...
Modules
Introduction

Numeric Functions & Logarithmic and Power functions
Calender Functions Set 1, Set 2
Complex Numbers Introduction & Important functions
Explore More...
Object Oriented Concepts
Class, Object and Members
Data Hiding and Object Printing
Inheritance, Subclass and super
Class method vs static method & Class or Static Variables
Explore More...
Exception Handling
Exception Handling
User-Defined Exceptions
Built-in Exceptions
Libraries and Functions
Timeit
Numpy Set 1, Set 2
Get and Post
import module & reload module
Collection Modules Deque, Namedtuple & Heap
Explore More...
Machine Learning with Python
Classifying data using Support Vector Machines(SVMs) in Python

K means Clustering
How to get synonyms/antonyms from NLTK WordNet in Python?
Explore More...
Misc
Sql using Python & MongoDB and Python
Json formatting & Python Virtual environment
Metaprogramming with Metaclasses in Python
Python Input Methods for Competitive Programming
Explore More...
Applications and Projects
Creating a proxy webserver Set 1, Set 2
Send Messsage to FB friend
Twitter Sentiment Analysis & Whatsapp using Python
Desktop Notifier & Junk File Organizer
Explore More...

pickle — Python object serialization

The pickle module is used for implementing binary protocols for serializing and de-serializing a Python object structure.

- Pickling: It is a process where a Python object hierarchy is converted into a byte stream.
- Unpickling: It is the inverse of Pickling process where a byte stream is converted into an object hierarchy.

Module Interface :

- dumps() – This function is called to serialize an object hierarchy.

- loads() – This function is called to de-serialize a data stream.

For more control over serialization and de-serialization, Pickler or an Unpickler objects are created respectively.

Constants provided by the pickle module :

1. **pickle.HIGHEST_PROTOCOL**
   This is an integer value representing the highest protocol version available. This is considered as the protocol value which is passed to the functions dump(), dumps().
2. **pickle.DEFAULT_PROTOCOL**
   This is an integer value representing the default protocol used for pickling whose value may be less than the value of highest protocol.

Functions provided by the pickle module :

1. **pickle.dump(obj, file, protocol = None, *, fix_imports = True)**
   This function is equivalent to Pickler(file, protocol).dump(obj). This is used to write a pickled representation of obj to the open file object file.

   The optional protocol argument is an integer that tells the pickler to use the given protocol.The supported protocols are 0 to HIGHEST_PROTOCOL. If not specified, the default is DEFAULT_PROTOCOL. If a negative number is specified, HIGHEST_PROTOCOL is selected.

   If fix_imports is true and protocol is less than 3, pickle will try to map the new Python 3 names to the old module names used in Python 2, so that the pickle data stream is readable with Python 2.

   Example :

```
# Python program to illustrate
# pickle.dump()
import pickle
from StringIO import StringIO

class SimpleObject(object):

    def __init__(self, name):
        self.name = name
        l = list(name)
        l.reverse()
        self.name_backwards = ''.join(l)
        return

data = []
data.append(SimpleObject('pickle'))
data.append(SimpleObject('cPickle'))
data.append(SimpleObject('last'))

# Simulate a file with StringIO
out_s = StringIO()

# Write to the stream
for o in data:
    print 'WRITING: %s (%s)' % (o.name, o.name_backwards)
```

```
    pickle.dump(o, out_s)
    out_s.flush()
```

Run on IDE

Output :

```
WRITING: pickle (elkcip)
WRITING: cPickle (elkciPc)
WRITING: last (tsal)
```

2. **pickle.dumps(obj, protocol = None, *, fix_imports = True)**
   This function returns the pickled representation of the object as a bytes object.

   Example :

```
# Python program to illustrate
#Picle.dumps()
import pickle

data = [ { 'a':'A', 'b':2, 'c':3.0 } ]
data_string = pickle.dumps(data)
print 'PICKLE:', data_string
```

Run on IDE

Output :

```
PICKLE: (lp0
(dp1
S'a'
p2
S'A'
p3
sS'c'
p4
F3.0
sS'b'
p5
I2
sa.
```

3. **pickle.load(file, *, fix_imports = True, encoding = “ASCII”, errors = “strict”)**
   This function is equivalent to Unpickler(file).load(). This function is used to read a pickled object representation from the open file object file and return the reconstituted object hierarchy specified.

   Example :

```
# Python program to illustrate
# pickle.load()
import pickle
from StringIO import StringIO
```

```
class SimpleObject(object):

    def __init__(self, name):
        self.name = name
        l = list(name)
        l.reverse()
        self.name_backwards = ''.join(l)
        return

data = []
data.append(SimpleObject('pickle'))
data.append(SimpleObject('cPickle'))
data.append(SimpleObject('last'))

# Simulate a file with StringIO
out_s = StringIO()


# Write to the stream
for o in data:
    print 'WRITING: %s (%s)' % (o.name, o.name_backwards)
    pickle.dump(o, out_s)
    out_s.flush()


# Set up a read-able stream
in_s = StringIO(out_s.getvalue())

# Read the data
while True:
    try:
        o = pickle.load(in_s)
    except EOFError:
        break
    else:
        print 'READ: %s (%s)' % (o.name, o.name_backwards)
```

Run on IDE

Output :

```
WRITING: pickle (elkcip)
WRITING: cPickle (elkciPc)
WRITING: last (tsal)
READ: pickle (elkcip)
READ: cPickle (elkciPc)
READ: last (tsal)
```

4. **pickle.loads(bytes_object, *, fix_imports = True, encoding = “ASCII”, errors = “strict”)**

   This function is used to read a pickled object representation from a bytes object and return the reconstituted object hierarchy specified.

   Example :

```
# Python program to illustrate
# pickle.loads()
import pickle
```

```
import pprint

data1 = [ { 'a':'A', 'b':2, 'c':3.0 } ]
print 'BEFORE:',
pprint.pprint(data1)

data1_string = pickle.dumps(data1)

data2 = pickle.loads(data1_string)
print 'AFTER:',
pprint.pprint(data2)

print 'SAME?:', (data1 is data2)
print 'EQUAL?:', (data1 == data2)
```

Run on IDE

Output :

```
BEFORE:[{'a': 'A', 'b': 2, 'c': 3.0}]
AFTER:[{'a': 'A', 'b': 2, 'c': 3.0}]
SAME?: False
EQUAL?: True
```

Exceptions provided by the pickle module :

1. **exception pickle.PickleError**
   This exception inherits Exception. It is the base class for all other exceptions raised in pickling.
2. **exception pickle.PicklingError**
   This exception inherits PickleError. This exception is raised when an unpicklable object is encountered by Pickler.
3. **exception pickle.UnpicklingError**
   This exception inherits PickleError. This exception is raised when there is a problem like data corruption or a security violation while unpickling an object.

Classes exported by the pickle module:

1. **class pickle.Pickler(file, protocol = None, *, fix_imports = True)**
   This class takes a binary file for writing a pickle data stream.
   1. **dump(obj) –** This function is used to write a pickled representation of obj to the open file object given in the constructor.
   2. **persistent_id(obj) –** If persistent_id() returns None, obj is pickled as usual. This does nothing by deafult and exists so that any subclass can override it.
   3. **Dispatch_table –** A pickler object's dispatch table is a mapping whose keys are classes and whose values are reduction functions.
      By default, a pickler object will not have a dispatch_table attribute, and it will instead use the global dispatch table managed by the copyreg module.

Example : The below code creates an instance of pickle.Pickler with a private dispatch table which handles the SomeClass class specially.

```
f = io.BytesIO()
p = pickle.Pickler(f)
p.dispatch_table = copyreg.dispatch_table.copy()
p.dispatch_table[SomeClass] = reduce_SomeClass
```

4. **Fast –** The fast mode disables the usage of memo and speeds up the pickling process by not generating superfluous PUT opcodes.

2. **class pickle.Unpickler(file, *, fix_imports = True, encoding = "ASCII", errors = "strict")**

This class takes a binary file for reading a pickle data stream.

1. **load() –** This function is used to read a pickled object representation from the open file object file and return the reconstituted object hierarchy specified.
2. **persistent_load(pid) –** This raises an UnpicklingError by default.
3. **find_class(module, name) –** This function imports module if required and returns the object called name from it, where the module and name arguments are str objects.

What can be pickled and unpickled?
The following types can be pickled :

- 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
- instances of such classes whose __dict__ or the result of calling __getstate__() is picklable

Pickling Class Instances :
This section explains the general mechanisms available to define, customize, and control how class instances are pickled and unpickled.
No additional code is needed to make instances picklable. By default, pickle will retrieve the class and the attributes of an instance via introspection.

Classes can alter the default behaviour by providing one or several special methods :

1. **object.__getnewargs_ex__()**
This method dictates the values passed to the __new__() method upon unpickling. The method must return a pair (args, kwargs) where args is a tuple of positional arguments and kwargs a dictionary of named arguments for constructing the object.

2. **object.__getnewargs__()**
   This method supports only positive arguments. It must return a tuple of arguments args which will be passed to the __new__() method upon unpickling.
3. **object.__getstate__()**
   If this method is defined by classes, it is called and the returned object is pickled as the contents for the instance, instead of the contents of the instance's dictionary.
4. **object.__setstate__(state)**
   If this method is defined by classes, it is called with the unpickled state. The pickled state must be a dictionary and its items are assigned to the new instance's dictionary.
5. **object.__reduce__()**
   The __reduce__() method takes no argument and shall return either a string or preferably a tuple.
6. **object.__reduce_ex__(protocol)**
   This method is similar to __reduce__ method. It takes a single integer argument. The main use for this method is to provide backwards-compatible reduce values for older Python releases.

Example : Handling Stateful Objects

This example shows how to modify pickling behavior for a class. The TextReader class opens a text file, and returns the line number and line contents each time its readline() method is called.

1. If a TextReader instance is pickled, all attributes except the file object member are saved.
2. When the instance is unpickled, the file is reopened, and reading resumes from the last location.

```
class TextReader:
    """Print and number lines in a text file."""

    def __init__(self, filename):
        self.filename = filename
        self.file = open(filename)
        self.lineno = 0

    def readline(self):
        self.lineno + = 1
        line = self.file.readline()
        if not line:
            return None
        if line.endswith('\n'):
            line = line[:-1]
        return "%i: %s" % (self.lineno, line)

    def __getstate__(self):
        # Copy the object's state from self.__dict__ which contains
        # all our instance attributes. Always use the dict.copy()
        # method to avoid modifying the original state.
        state = self.__dict__.copy()
        # Remove the unpicklable entries.
        del state['file']
        return state

    def __setstate__(self, state):
        # Restore instance attributes (i.e., filename and lineno).
        self.__dict__.update(state)
        # Restore the previously opened file's state. To do so, we need to
        # reopen it and read from it until the line count is restored.
        file = open(self.filename)
        for _ in range(self.lineno):
            file.readline()
```

```
        # Finally, save the file.
        self.file = file

reader = TextReader("hello.txt")
print(reader.readline())
print(reader.readline())
new_reader = pickle.loads(pickle.dumps(reader))
print(new_reader.readline())
```

Run on IDE

Output :

```
'1: Hello world!'
'2: I am line number two.'
'3: Goodbye!'
```

This article is contributed by **Aditi Gupta**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Python Python-Library

Recommended Posts:

Understanding Python Pickling with example
Text Analysis in Python 3
Tokenize text using NLTK in python
Textwrap – Text wrapping and filling in Python
copyreg — Register pickle support functions

(Login to Rate and Mark)

2 Average Difficulty : **2/5.0**
Based on **1** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!