Serialization and Closure

Session 10

Session Overview

In this session, you will be able to:

- Describe the concept of serialization in Python
- Describe the uses of `pickle` module
- Outline the procedure of pickling Python objects
- Describe how to implement closures in Python
- Describe the concept of sets in Python
- Identify the different operations on sets in Python

Data Serialization in Python

The process of converting structured data or object state in a way that its original structure can be recovered is called as data serialization.

Replica of the original object can be generated by a developer while restoring the resultant format to its original form.

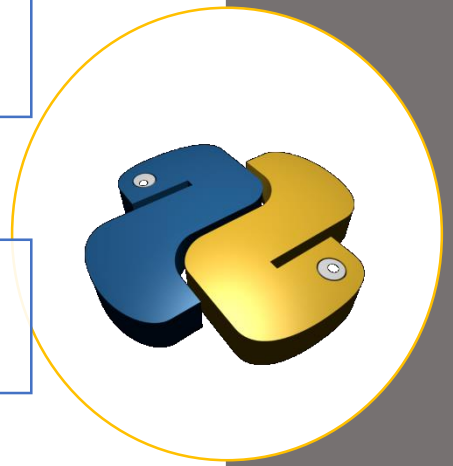
Pickle Module (1-2)

In Python, serialization is possible through the pickle module.

It is the standard library for serializing data and objects in Python through binary protocols.

Modules for serializing, two widely used data formats:

- JavaScript Object Notation (JSON)
- XML-encoded property lists (plistlib)



Pickle Module (2-2)

Types of data structures that pickle can store:

- Native data types
- Sets, dictionaries, tuples, and lists holding any mix of native datatypes
- Sets, dictionaries, tuples, and lists holding any mix of sets, tuples, dictionaries, and lists holding any blend of native datatypes
- Instances, classes, and functions

Pickling and its Protocol

- Data format of pickle module is specific to Python.
- Five protocols that the module uses for pickling:

Module Interface (1-2)

The module provides constants, functions, and exceptions.

Following are the two constants:

- `pickle.HIGHEST_PROTOCOL`
- `pickle.DEFAULT_PROTOCOL`

Module Interface (2-2)

Different functions of the pickle module:

- `pickle.dump()`
- `pickle.dumps()`
- `pickle.load()`
- `pickle.loads()`

Creating and Fetching Pickle Data from a File (1-5)

Python script to be executed in shell X:

Code Snippet 1:

```
>>> shell = 'X'
>>> book = {}
>>> book['title'] = 'Python Book'
>>> book['page_link'] = 'http://example_link'
>>> book['comment_link'] =
'http://example_comment_link'
>>> book['id'] = b'\xA1\xED\xC1\x37'
>>> book['tags'] = ('Python', 'Coding', 'Technical')
>>> book['published'] = True
>>> import time
>>> book['published_time'] = time.strptime('Sun Sep
10 23:18:32 2017')
```

Creating and Fetching Pickle Data from a File (2-5)

Script to save the code in Code Snippet 1 as a pickle file:

Code Snippet 2:

```
>>> import pickle
>>> with open('book.pickle',
'wb') as file:
            pickle.dump(book,
file )
```

Creating and Fetching Pickle Data from a File (3-5)

How to open the pickled data from another shell, Y:

Code Snippet 3:

```
>>> import pickle
>>> with open('book.pickle', 'rb') as file:
        file_loaded = pickle.load(file)
>>> file_loaded
{'title': 'Python Book', 'page_link':
'http://example_link', 'comment_link':
'http://example_comment_link', 'id': b'\xa1\xed\xc17',
'tags': ('Python', 'Coding', 'Technical'), 'published':
True, 'published_time': time.struct_time(tm_year=2017,
tm_mon=9, tm_mday=10, tm_hour=23, tm_min=18, tm_sec=32,
tm_wday=6, tm_yday=253, tm_isdst=-1)}
```

Creating and Fetching Pickle Data from a File (4-5)

Switching to shell X, Code Snippet 4 shows the difference between equal and identical objects:

Code Snippet 4:

```
>>> with open('book.pickle', 'rb') as file:
        book2 = pickle.load(file)
>>> book2
{'title': 'Python Book', 'page_link':
'http://example_link', 'comment_link':
'http://example_comment_link', 'id': b'\xa1\xed\xc17',
'tags': ('Python', 'Coding', 'Technical'), 'published':
True, 'published_time': time.struct_time(tm_year=2017,
tm_mon=9, tm_mday=10, tm_hour=23, tm_min=18, tm_sec=32,
tm_wday=6, tm_yday=253, tm_isdst=-1)}
```

Creating and Fetching Pickle Data from a File (5-5)

```
>>> book
{'title': 'Python Book', 'page_link':
'http://example_link', 'comment_link':
'http://example_comment_link', 'id': b'\xa1\xed\xc17',
'tags': ('Python', 'Coding', 'Technical'), 'published':
True, 'published_time': time.struct_time(tm_year=2017,
tm_mon=9, tm_mday=10, tm_hour=23, tm_min=18, tm_sec=32,
tm_wday=6, tm_yday=253, tm_isdst=-1)}
>>> if book == book2 :
        print ('True, they are the same')
    else :
        print ('They are not same')
>>> if book is book2 :
        print ('book2 is book')
    else:
        print ('book2 is not book')
```

```
Output:
True, they are
the same
book2 is not
book
```

Serializing or Pickling Data in Memory (1-2)

Serialize data in memory in shell X using pickle module:

Code Snippet:

```
>>> m = pickle.dumps(book)
>>> m
b'\x80\x03}q\x00(X\x05\x00\x00\x00titleq\x01X\x0b\x00\x00\x00Pytho
n
Bookq\x02X\t\x00\x00\x00page_linkq\x03X\x13\x00\x00\x00http://exam
ple_linkq\x04X\x0c\x00\x00\x00comment_linkq\x05X\x1b\x00\x00\x00ht
tp://example_comment_linkq\x06X\x02\x00\x00\x00idq\x07C\x04\xa1\xe
d\xc17q\x08X\x04\x00\x00\x00tagsq\tX\x06\x00\x00\x00Pythonq\nX\x06
\x00\x00\x00Codingq\x0bX\t\x00\x00\x00Technicalq\x0c\x87q\rX\t\x00
\x00\x00publishedq\x0e\x88X\x0e\x00\x00\x00published_timeq\x0fctim
e\nstruct_time\nq\x10(M\xe1\x07K\tK\nK\x17K\x12K
K\x06K\xfdJ\xff\xff\xff\xfftq\x11}q\x12(X\x07\x00\x00\x00tm_zoneq\
x13NX\t\x00\x00\x00tm_gmtoffq\x14Nu\x86q\x15Rq\x16u.'
```

Serializing or Pickling Data in Memory (2-2)

Code Snippet:

```
>>> type(m)
<class 'bytes'>
>>> book3 = pickle.loads(m)
>>> if book3 == book :
        print ("pickle loaded from the memory is same")
    else:
        print ("it is different")
```

Output:

```
pickle loaded from the memory is same
```

Advantages of Pickle Module

Benefits of pickle module:

- Efficient Serialization
- Object Sharing
- Support for User-defined Classes and their Instances

Introduction to Closures and its Significance

It is important to understand the concept of nested functions and non-local variables for knowing closures.

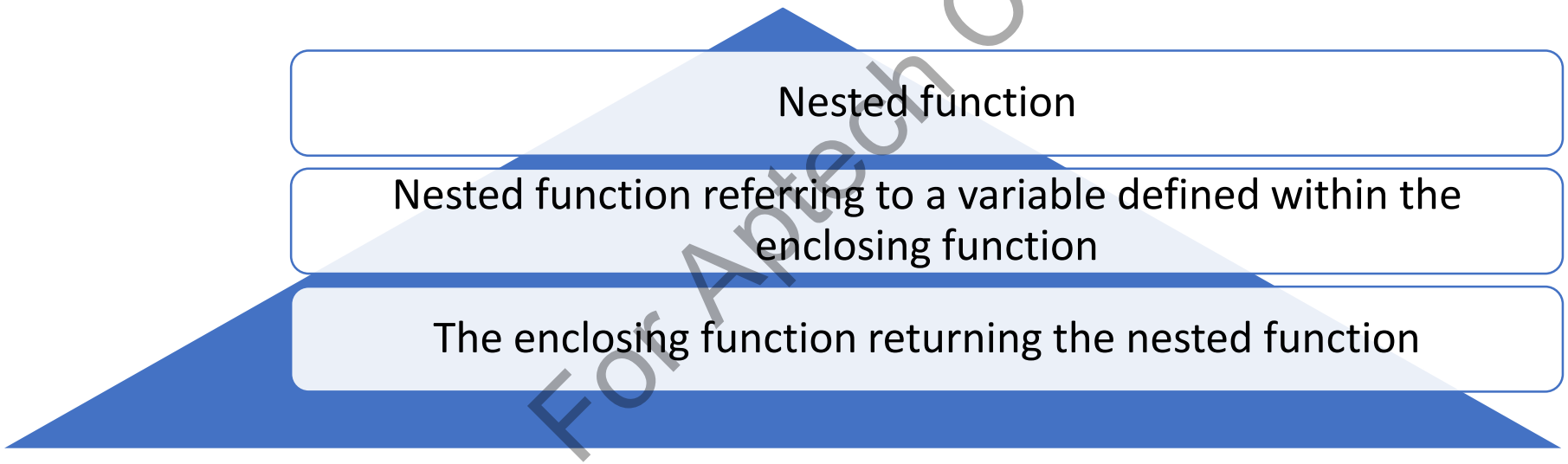
Nested function is a function defined within another function.

It is capable of accessing and handling variables of the enclosing scope.

Closure can be defined as the mechanism by which the data remains associated with the code despite the execution of original functions is over.

Creating Closures

- Developers can create closures via function calls in Python.
- Criteria for creating a closure in Python:



Using the nonlocal Keyword in Closures

Closure function taking no argument for adding all numbers until a specific range:

Code Snippet:

```
>>> def closuref(max):
      num = 0
      def nestedf():
        nonlocal num
        for k in range(max+1):
            num += k
        print("Total = %d" % num)
      return nestedf
>>> gett = closuref(5)
>>> gett()
```

Output:

```
>>> Total = 15
```

The `__closure__` Attribute and `Cell` Objects (1-2)

Following Code Snippet shows how to use this attribute:

Code Snippet:

```
>>> def start(start_at):
        def increment(increment_by):
                return start_at + increment_by
        return increment
>>> first = start (12)
>>> second = start (100)

>>> print((type(start))
<class 'function'>
```

The `__closure__` Attribute and `Cell` Objects (2-2)

```
>>> print(first.__closure__)
<cell at 0x102b94a68: int object at 0x10028bd80>
>>> print(type(first.__closure__[0]))
<class 'cell'>
>>> print(first.__closure__[0].cell_contents)
>>> print(second.__closure__)
<cell at 0x102b94ca8: int object at 0x10028c880>

>>> print(type(second.__closure__[0]))
<class 'cell'>
>>> print(second.__closure__[0].cell_contents)
second.__closure__[0].cell_contents=100
```

Output:

```
12
```

Using Lambda

Lambda can be used for fulfilling the goal of a nested function. Following Code Snippet shows how to do so:

Code Snippet:

```
>>> def start(start_at):
                return lambda increment:
start_at+increment

>>> first = start(12)
>>> second = start (100)
>>> print (first(3))
>>> print (second(4))
```

Output :

```
15
104
```

Closure Uses

- Closures act as callback functions and supports hiding data in an application.
- This facilitates in reducing the usage of global variables.
- They are also capable of offering an object oriented solution.
- When an application has only a few functions, closures offers efficient solutions.

Introduction to Sets in Python

- Set in Python is a data structure, which is analogous to the mathematical concept of sets.
- It can hold a variety of elements.
- Developer can insert and remove elements of a set, iterate over them, and can perform standard set operations.
- These operations include difference, union, and intersection.
- One can also find out whether an element belongs to a specific set.

Creating Sets

Define a set using the set() function:

Code Snippet:

```
>>> a = set('qwerty')
>>> a
>>> A = set('banana')
>>> A
```

Output :

```
{'t', 'q', 'r', 'y', 'e', 'w'}
{'n', 'b', 'a'}
```

Operations on Sets

With the `len()` function, a developer can easily fetch the number of elements in the set.

The developer can also iterate over the set elements in an undefined order with the help of the for loop.

Using the `in` keyword, a developer can check whether an element exists in a set.

To know whether an element is not in a particular set, the `not in` keyword is used.

An element to a set can be added by using the built-in `add()` method.

Summary (1-5)

- Python implements pass by object or by object reference for passing arguments.
- Immutable arguments implement pass by value mode in Python.
- Mutable arguments implement pass by reference.
- The single asterisk syntax `(*args)` allows specifying a list of non-keyworded arguments of variable length.
- The double asterisk syntax `(**kwargs)` allows passing a list of keyworded arguments of variable length. It handles named arguments.

Summary (2-5)

- `*args` and `**kwargs` always come after formal arguments in a function header.
- A regular expression is a special string that represents a search pattern for fetching information from the targeted string or file data.
- REs can have ordinary and special characters, of which the latter ones do not match themselves and are called metacharacters.

Summary (3-5)

- Some of the most useful special characters are $, ., *, +, ?, |, (...), [ ], { }, and \.
- The re module offers functions for running regular expressions as strings.
- Python compiles regular expressions into `RegexObject` instances that come with methods for operations such as substituting strings or searching for pattern matches.

Summary (4-5)

- The `match()` method looks for a match only at the start of the string, while `search()` looks for it anywhere in the string.
- The `findall()` method allows iterating over the lines and returns a list of all matches at once.
- The `split()` method divides a string by the pattern's occurrences.

Summary (5-5)

- The sub() method searches for all occurrences of a pattern and replace them with a new sub string.
- Some of the most useful methods of a MatchObject instance are group(), span(), start(), and end().