Lab-4 : File handling operations and miscellaneous concepts in Python.

#
#
1. **Aim :** To implement file handling programs and study other miscellaneous functions / libraries in Python.

2. **Objective** :

In this lab, the objective is to learn about various file handling operations in python. This lab also aims to develop understanding about various miscellaneous functions / libraries available in Python. Understanding of such libraries helps to write the code in a faster and optimized way.

3. **Description:**

Implementation of the programs for the below mentioned concepts should be done in this lab :

- File handling : Open, Read, Write various files in python.
- Collections
- itertools
- functools
- Lambda

4. **Examples:**

File Handling :

In Python, we use the open() method to open files.

Example :

```
# open file in current directory
file1 = open("test.txt")
```

Different modes of opening file :
r (read), w (write), x(exclusive creation), a (append)

Reading a file :

```
# open a file
file1 = open("test.txt", "r")

# read the file
read_content = file1.read()
print(read_content)
```

Writing to a file :

```
with open(test2.txt', 'w') as file2:

    # write contents to the test2.txt file
    file2.write('Programming is Fun.')
    fil2.write('Programiz for beginners')
```

Collections :

Python programming language has four collection data types- list, tuple, sets and dictionary. But python also comes with a built-in module known as collections which has specialized data structures which basically covers for the shortcomings of the four data types.

Collections module in python implements specialized data structures which provide alternative to python’s built-in container data types. Following are the specialized data structures in collections module.

namedtuple( ) : It returns a tuple with a named entry, which means there will be a name assigned to each value in the tuple.

deque : deque pronounced as ‘deck’ is an optimized list to perform insertion and deletion easily.

Chainmap : It is a dictionary like class which is able to make a single view of multiple mappings.

Counter : It is a dictionary subclass which is used to count hashable objects.

OrderedDict : It is a dictionary subclass which remembers the order in which the entries were added.

defaultdict : It is a dictionary subclass which calls a factory function to supply missing values.

UserDict : This class acts as a wrapper around dictionary objects. The need for this class came from the necessity to subclass directly from dict.

UserList : This class acts like a wrapper around the list objects. It is a useful base class for other list like classes which can inherit from them and override the existing methods or even add a fewer new ones as well.

Example :

```
from collections import namedtuple
a = namedtuple('courses' , 'name , tech')
s = a('data science' , 'python')
print(s)
```

```
#the output will be courses(name='python' , tech='python')
```

```
#creating a deque
from collections import deque

a = ['d' , 'u' , 'r' , 'e' , 'k']
a1 = deque(a)
print(a1)
#the output will be deque([ 'd' , 'u' , 'r' , 'e' , 'k' ])

from collections import ChainMap
a = { 1: 'edureka' , 2: 'python'}
b = {3: 'data science' , 4: 'Machine learning'}
c = ChainMap(a,b)
print(c)
#the output will be ChainMap[{1: 'edureka' , 2: 'python'} , {3: 'data science' , 4: 'Machine
learning'}]
```

itertools

The itertools module is a standard library in Python that provides fast and efficient tools for working with iterators. An iterator is an object that produces a stream of values, one at a time, when iterated over.

The itertools module provides a number of functions for working with iterators. Some examples are :

count(): Generates an iterator that produces an infinite sequence of numbers, starting from a given value and incrementing by a specified step.

cycle(): Generates an iterator that repeats a sequence of elements indefinitely.

repeat(): Generates an iterator that produces a given element a specified number of times.

chain(): Takes a number of iterators as arguments and returns a new iterator that produces the elements of each iterator in turn.

compress(): Filters elements from an iterator based on a list of boolean values.

Combinatoric iterators : The recursive generators that are used to simplify combinatorial constructs such as permutations, combinations, and Cartesian products are called combinatoric iterators.

Example :

```
# import the product function from itertools module
from itertools import product

print("The cartesian product using repeat:")
print(list(product([1, 2], repeat = 2)))
print()

print("The cartesian product of the containers:")
print(list(product(['go', 'adam', 'go'], '2')))
print()

print("The cartesian product of the containers:")
print(list(product('AB', [3, 4])))
```

Output :

```
The cartesian product using repeat:
[(1, 1), (1, 2), (2, 1), (2, 2)]

The cartesian product of the containers:
[(go, '2'), ('adam', '2'), ('go', '2')]

The cartesian product of the containers:
[('A', 3), ('A', 4), ('B', 3), ('B', 4)]
```

functools

The functools module is a standard library in Python that provides tools for working with functions. It provides a number of functions that allow you to modify the behavior of other functions, as well as decorators and higher-order functions.

Example of how you can use the partial() function from the functools module to create a new function with some of the arguments of another function fixed:

```
import functools

# Define a function that takes two arguments and returns their product
def multiply(x, y):
    return x * y

# Create a new function that always multiplies by 3
triple = functools.partial(multiply, 3)

# Test the new function
print(triple(4))  # Output: 12
print(triple(5))  # Output: 15
```

Lambda :

A lambda function is a small anonymous function without a name. It can be used when you need a function for a short period of time, and don't want to define a function using the def keyword.

example of how you can define and use a lambda function in Python:

```
# Define a lambda function that adds two numbers
add = lambda x, y: x + y

# Use the lambda function
result = add(3, 4)
print(result)  # Output: 7
```

Lambda functions are often used as arguments to other functions, such as the map() and filter() built-in functions. For example:

```
# Use the map() function to apply a lambda function to a list of numbers
numbers = [1, 2, 3, 4]
result = map(lambda x: x ** 2, numbers)
print(list(result))  # Output: [1, 4, 9, 16]

# Use the filter() function to select elements from a list based on a lambda function
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = filter(lambda x: x % 2 == 0, numbers)
print(list(result))  # Output: [2, 4, 6, 8, 10]
```

JSON in Python :

Python has a built-in package called json, which can be used to work with JSON data.

Example :

```
import json

# some JSON:
x = '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
```

```
print(y["age"])
```

You can convert Python objects of the following types, into JSON strings:

- dict
- list
- tuple
- string
- int
- float
- True
- False
- None

```
import json

print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
```

5. **Exercise:**

1. Write a Python program to count the frequency of words in a file.
2. Write a Python program to write a list to a file.
3. Write a Python program to generate 26 text files named A.txt, B.txt, and so on up to Z.txt.
4. Write a Python program to convert Python objects into JSON strings. Print all the values.
5. Write a Python program to square and cube every number in a given list of integers using Lambda.
6. Write a Python program to find palindromes in a given list of strings using Lambda.
7. Write a Python program to create a Cartesian product of two or more given lists using itertools.
8. Write a Python program to generate all possible permutations of n different objects.

6. References:

1. https://docs.python.org/3/tutorial/index.html
2. https://www.w3schools.com/python/
3. https://jupyter.org/
4. https://www.python.org/