Module-2

Lists, Dictionaries and Structuring Data

1. Discuss list and dictionary data structure with example for each.

2. What is list and dictionary? What are the differences between List & Dictionary?

3. Explain the methods that are used to delete items from the list

4. Explain negative indexing, slicing, index(),append(),remove(),pop(),insert() & sort().

5. Explain with a programming example to each:
   (ii) get ()
   (iii)setdefault()

6. Explain the following methods in list with an examples:

   (i)len() (ii)sum() (iii)max() (iv)min()

7. Develop suitable Python programs with nested lists to explain copy.copy( ) and copy.deepcopy( ) methods.

8. Explain the use of in operator and not in operators in list with suitable examples.

Programs

1. Read a multi-digit number (as chars) from the console. Develop a program to print the frequency of each digit with suitable message.

2. Develop a program to find mean, variance and standard deviation.

3. Write a python program to accept n numbers and store them in a list. Then print the list without ODD numbers in it.

4. Develop a python program to swap cases of a given string.

   Input: Java

   Output: jAVA

1. **Discuss list and dictionary data structure with example for each.**

List: A list is an ordered data structure with elements separated by a comma and enclosed within square brackets. A list is a value that contains multiple values in an ordered sequence.

A list value looks like this: ['cat', 'bat', 'rat', 'elephant'].

Just as string values are typed with quote characters to mark where the string begins and ends, a list begins with an opening square bracket and ends with a closing square bracket, []. Values inside the list are also called items. Items are separated with commas.
Examples: [1, 2, 3], ['hello', 3.1415, True, None, 42],
spam = ['cat', 'bat', 'rat', 'elephant']

Dictionary: Dictionary is a collection of many values. The dictionary is an unordered collection that contains key: value pairs separated by commas inside curly brackets
myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
This dictionary's keys are 'size', 'color', and 'disposition'. The values for these keys are 'fat', 'gray', and 'loud', respectively.
Dictionaries can still use integer values as keys, just like lists use integers for indexes, but they do not have to start at 0 and can be any number.
spam = {12345: 'Luggage Combination', 42: 'The Answer'}

2. **What are the differences between List & Dictionary?**

List	**Dictionary**
A list is an ordered data structure	Items in dictionaries are unordered
List is a mutable	Dictionary is a mutable
Lists are denoted by [ ] square brackets	Dictionary is denoted by { } Curly braces
Ex: ['cat', 'bat', 'rat', 'elephant'], [1,2,3]	myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
Slicing can be done	Slicing cannot be done
Lists can contain duplicate elements	Cannot contain duplicate keys, but can contain du[plicate values

```
>>> spam = ['cats', 'dogs', 'moose']
>>> bacon = ['dogs', 'moose', 'cats']
>>> spam == bacon
```
False
```
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == ham
```
True

3. **Explain the methods that are used to delete items from the list.**

Removing Values from Lists with del Statements

The del statement will delete values at an index in a list. All of the values in the list after the deleted value will be moved up one index. For example

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

Removing Values from Lists with remove()

The remove() method is passed the value to be removed from the list it is called on.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

4. **Explain negative indexing, slicing, index(),append(),remove(),pop(),insert() & sort().**

Negative Indexing: While indexes start at 0 and go up, you can also use negative integers for the index. The integer value -1 refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
```

```
>>> spam[-3]
'bat'
```

Slicing: A slice can get several values from a list, in the form of a new list. A slice is typed between square brackets, like an index, but it has two integers separated by a colon.

In a slice, the first integer is the index where the slice starts. The second integer is the index where the slice ends. A slice goes up to, but will not include, the value at the second index. A slice evaluates to a new list value.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

index(): In a list ['cat', 'bat', 'rat', 'elephant'] stored in a variable named spam. The Python code spam[0] would evaluate to 'cat', and spam[1] would evaluate to 'bat', and so on. The integer inside the square brackets that follows the list is called an index. The first value in the list is at index 0, the second value is at index 1, and the third value is at index 2, and so on.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[1]
'bat'
>>> spam[2]
'rat'
>>> spam[3]
'elephant
```

remove(): The remove() method is passed the value to be removed from the list it is called on.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

pop(): Removes the element at the specified position.

```
fruits = ['apple', 'banana', 'cherry']
fruits.pop(1)
print(fruits)
['apple',  'cherry']
```

insert(): The insert() method can insert a value at any index in the list. The first argument to insert() is the index for the new value, and the second argument is the new value to be inserted.

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

sort(): Lists of number values or lists of strings can be sorted with the sort() method. For example, enter the following into the interactive shell**:**

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

5. Explain with a programming example to each:
(i) get ()
(ii)setdefault()
get () Method
It’s tedious to check whether a key exists in a dictionary before accessing that key’s value. Fortunately, dictionaries have a get() method that takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist**.**

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs
```

The setdefault() Method

You'll often have to set a value in a dictionary for a certain key only if that key does not already have a value. The code looks something like this:

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
spam['color'] = 'black'
```

The setdefault() method offers a way to do this in one line of code. The first argument passed to the method is the key to check for, and the second argument is the value to set at that key if the key does not exist. If the key does exist, the setdefault() method returns the key's value.

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

6. **Explain the following methods in list with an examples:**

(i)len() (ii)sum() (iii)max() (iv)min()

(i)len() : The len() function will return the number of values that are in a list value passed to it, just like it can count the number of characters in a string value.

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

(ii)sum(): Python's built-in function sum() is an efficient and Pythonic way to sum a list of numeric values.

```
numbers = [1,2,3,4,5,1,4,5]
Sum = sum(numbers)
print(Sum)

Output: 25
```

(iii)max(): The max() function returns the item with the highest value, or the item with the highest value in a list.

```
a = [1, 5, 3, 9]
x = max(a)

print(x)

output: 9
```

(iv)min(): The min() function returns the item with the lowest value, or the item with the lowest value in an iterable.
If the values are strings, an alphabetically comparison is done.

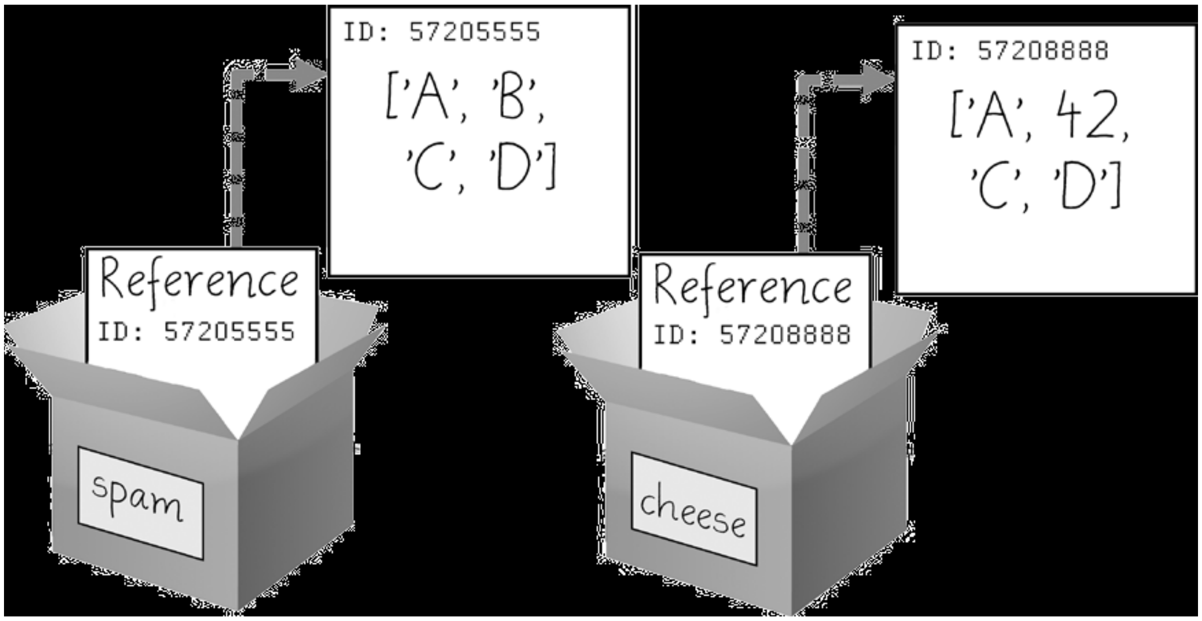
```
a = [1, 5, 3, 9]
x = min(a)

print(x)
output: 1
```

7. **Develop suitable Python programs with nested lists to explain copy.copy() and copy.deepcopy( ) methods.**

If the function modifies the list or dictionary that is passed, you may not want these changes in the original list or dictionary value. For this, Python provides a module named copy that provides both the copy () and deepcopy() functions. The first of these, copy.copy(), can be used to make a duplicate copy of a mutable value like a list or dictionary, not just a copy of a reference.

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

Now the spam and cheese variables refer to separate lists, which is why only the list in cheese is modified when you assign 42 at index 7. As you can see in Figure , the reference ID numbers are no longer the same for both variables because the variables refer to independent list.



If the list you need to copy contains lists, then use the copy.deepcopy() function instead of copy.copy(). The deepcopy() function will copy these inner lists as well.

8. **Explain the use of in operator and not in operators in list with suitable examples.**

The in and not in Operators

You can determine whether a value is or isn't in a list with the in and not in Operators.

Like other operators, in and not in are used in expressions and connect two values: a value to look for in a list and the list where it may be found.

These expressions will evaluate to a Boolean value.

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
print('Enter a pet name:')
name = input()
if name not in myPets:
        print('I do not have a pet named ' + name)
else:
        print(name + ' is my pet.')
```

The output may look something like this:

```
Enter a pet name:
Footfoot
I do not have a pet named Footfoot
```

3. Write a python program to accept n numbers and store them in a list. Then print the list without ODD numbers in it.

```
lst = [ ]
n = int(input("Enter number of elements : "))
for i in range(0, n):
    ele = int(input())
    lst.append(ele)
print(lst)
for i in lst:
    if i % 2 != 0:
        lst.remove(i)
print("List after removing ODD numbers:")
print(lst)
```

```
Output: Enter number of elements: 5
1
2
3
4
5
[1, 2, 3, 4, 5]
List after removing ODD numbers:
[2, 4]
```

4. Develop a python program to swap cases of a given string.
Input: Java
Output: jAVA

```
string = "Java"

# prints after swapping all cases
print(string.swapcase())
```

```
Input: Java
Output: jAVA
```