DAY-2

1.Lists:

Lists are used to store multiple items in a single variable.Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

- Access Items:

List items are indexed and you can access them by referring to the index number:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Output: banana

- Change Item Value:

To change the value of a specific item, refer to the index number:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Output: ["apple", "blackcurrant", "cherry"]

- Append Items:

To add an item to the end of the list, use the append() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

output :["apple", "banana", "cherry","orange"]

- Remove Specified Item:

The remove() method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)   output:["apple", "cherry"]
```

- Loop Through a List

Print all items in the list,one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Output : apple
Banana
Cherry

2.TUPLES:

- Tuples are used to store multiple items in a single variable.
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.

Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

output: (‘apple’,’banana’,’cherry’)

- Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Output: banana

- Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

```
fruits = ("apple", "banana", "cherry")
print(fruits)
```

Output :("apple", "banana", "cherry")

- Loop Through a Tuple

You can loop through the tuple items by using a for loop.

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
 print(x)
```

Output: apple
banana
cherry

3.Dictionary

Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.Dictionaries are written with curly brackets, and have keys and values:

Create and print a dictionary:

```
thisdict = { "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}print(thisdict)
```

Output :'brand': 'Ford', 'model': 'Mustang','year': 1964

- Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

Output: Mustang

- Change Values:

Change the "year" to 2018:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

- Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

EX:

```
thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
```

```
thisdict["color"] = "red"
print(thisdict)
```

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

- Removing Items:

The pop() method removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Output: {'brand': 'Ford', 'year': 1964}

4.SET:

Sets are used to store multiple items in a single variable.
Set *items* are unchangeable, but you can remove items and add new items.
A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
Sets are written with curly brackets.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Output: {'banana', 'cherry', 'apple'}

5.List Comprehensions:

List comprehensions provide a concise way to create lists. They consist of brackets containing an expression followed by a for clause, and then zero or more for or if clauses. The result will be a new list resulting from evaluating the expression in the context of the for and if clauses.

Syntax:

[expression for item in iterable if condition]

Creating a list of squares from 0 to 9

squares = [x**2 for x in range(10)]

print(squares)

Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

6.**Lambda Functions:**

Lambda functions are small anonymous functions defined with the lambda keyword. They can have any number of arguments but only one expression. They are often used in places where you need a small function for a short period.

Syntax: lambda arguments: expression

ex:

A lambda function that adds 10 to its input

```
add_ten = lambda x: x + 10

print(add_ten(5))
```

Output: 15

7.Exception Handling:

Use try to execute code that might fail, and except to handle errors.

Syntax:

try:

    # Code that might raise an exception

except ExceptionType:

    # Code to handle the exception

Ex:

try: result = 10 / 0

except ZeroDivisionError: print("Cannot divide by zero!")