Week 5:

1. **For Loops:**
   First of all, we have to understand what is loop. Loop is basically the technique using which we can repeat the lines of code. It makes the task easy when we have to execute 1 statement 40 or 50 times because rather than writing 50 lines, we can execute those lines in 1 or 2 statement. It has starting and ending value so until loop will be executed statement will also executed. Because we emerge statement in loop.
   There are **two** types of loops in python:
   I. *For Loop:*
   II. *While Loop*

   For Loops is very simple and easy to use. We use Keyword **for** then we use a **variable** and Keyword **in** is use after variable and last, we write name of **list/variable**(on which we want to execute loop). Then like if statement we use colon :.

   - **for** (name of **variable** we create specifically for loop) **in list1**:

   Following code show the proper syntax and example:

```
list = [1,2,3,4,5]
for items in list:
    print(items)
```

   This is the code of for loop that will iterate over list and will print all items of list in new line.

   Result will be:

   1

   2

   3

   4

   5

2. **For loop Break Keywords:**
   Loop executes the statement till all the elements or values in list or values are finished. But there is a one method through which we can terminate loop.

   Break keyword is used and it will break or terminate the loop and rest of statement s of code will be executed.

Following lines of code explains it well:

```
list = [1,2,3,4,5]
for items in list:
    print(items)
    if items==3:
        break
```

So, here we executed loop on a list and gave condition that when loop reaches 3 in the list, break or terminate it.

Result:

1

2

3

3. **For loop Continue Keyword:**

   As we have seen in the previous video break end the loop. Continue statement is used to actually skip the specific item/value of list or variable. Like, that skip value will not be printed.

   This is shown by the following code:

```
list = [1,2,3,4,5]
for items in list:
    if items==3:
        continue
    print(items)
```

It will skip 3 and print all items of list.

Result:

1

2

4

5

4. **For Loop – In practice:**

   We have syntax of for to repeat lists element now we will see that how can we print whatever we want and how much times we want.

   So, syntax for that will be

   for (create a variable) in range(how many times you want repetition):

Code:

```
for a in range(5):
    print(a,"I am Usman", sep="-")
```

Result:
0-I am Usman
1-I am Usman
2-I am Usman
3-I am Usman
4-I am Usman

It starts from 0 and go up to 4 and print the message 5 times.

```
for a in range(10):
    print(a)
```

As I said this loop will start from 0 go up to 10 because range function starts from 0.

Result:

0

1

2

3

4

5

6

7

8

9

- **But There is a one thing through which we can control range to starts from where ever we want.**

```
for a in range(1,10):
    print(a)
```

But it will be finished at 9. So, result will be:

1

2

3

4

5

6

7

8

9

- **We can also add slicing feature of list in which it will skip the numbers of item or statements according to our call.**

```
for a in range(1,10,3):
    print(a)
```

We gave the skipping value 3 so it will skip 3 numbers and will print the other number.So, the result will be :

1

4

7

- **We can also skip items in backward direction by just putting – sign with the skipping steps.**

```
for numbers in range(21,1,-2):
    print(numbers)
```

Result:

21

19

17

15

13

11

9

7

5

3

- **If we give a variable having single string in for loop it will print all the characters of the strings and will give it as the result.**
  LIKE:

```
Country = "Pakistan"
for char in Country:
    print(char)
```

Result:
P
a
k
i
s
t
a
n

- **If we variable more than one string it will become tuple and will return one string each time when loop will execute it.**

```
Countries = "Pakistan","America","China"
for country in Countries:
    print(country)
```

Result:

Pakistan

America

China

5. **Nested for loop – In practice:**
   Nesting a for loop within the body of another for loop is known as nested for loop. In nested for 1 is inner for loop and other one is outer for loop**.**

   Syntax is shown in the following code:

```
for numbers in range(1,4):
    print(numbers)
    for character in "Hey":
        print(numbers,character)
```

Its result will be pretty long which will be:

1

1 H

1 e

1 y

2

2 H

2 e

2 y

3

3 H

3 e

3 y

So, the thing that happens is when one first loop executed it prints 1 then it enters in the second loop which was actually nested and second loop executed completely then it leaves the inner loop and go in outer loop prints 2 and comeback in the second loop and execute it completely. It happens for 3rd time too .

- **Code for printing any table:**

```
tablenumber = int(input("Enter ant number"))
for number in range(1,11):
    print(f"{tablenumber} * {number} = {tablenumber*number}" )
```

Result will be :

10 * 1 = 10
10 * 2 = 20
10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100

6. **Nested Loop – In practice 2:**

So, we have learned printing 1 table. With the help of nested loop, we can print more than 2 tables.

Code:

```
table = int(input("Enter any number please "))
for table in range(table,table+3):
    for numbers in range(1,11):
        print(f"{table}  *  {numbers}  = {table*numbers}")
```

Code Explanation:

Firstly, it will ask for a number after it first loop which is outer loop will be executed. We set the range from that given number to two numbers ahead. Then inner loop will be executed and table of first number will be printed then control will go back to outer loop and table of 2nd number which be the next number of previous numbers will be executed and same happens with the 3rd number.

So, Result will be:

Enter any number please 4

4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60

7. **Type casting:**
   Typecasting is changing data type of element or variable/input.
   Type casting is important because when we take some input from the user it is always in string data type so, we have to first convert it from string to integer if it is a number or float if it is a decimal number or Boolean if it is true or false.
   Syntax:

```
a = int(1.0)
print(type(a))
```

   Result:
   <class 'int'>
   I changed float value to integer by type casting.

8. **Type casting – In practice:**
   We can check the data type of variable and inputs by the type function. Like

```
a = 10
print(type(a))
b = str(a)
print(type(b))
```

   Result:
   <class 'int'>
   <class 'str'>
   We changed the fata type of a and store it in a variable and changes its data type.

```
a = (input("Enter any number :"))
b = (input("Enter nay number :"))
print(type(a))
print(type(b))
```

   We will put integers but it will return the str data type.
   Enter any number :2
   Enter nay number :2
   <class 'str'>
   <class 'str'>
   Type casting is important because when we take some input from the user it is always in string data type so, we have to first convert it from string to integer if it is a number or float if it is a decimal number or Boolean if it is true or false.

```
a = input("Enter any number :")
b = input("Enter nay number :")
c = a + b
print(c)
# It take it as string and concatenates it Here we need typecasting because we
have convert it from string to integer
a = int(input("Enter any number :"))
b = int(input("Enter nay number :"))
```

```
c = a+b
print(c)
```

Result:
Enter any number :12
Enter nay number :10
1210
Enter any number :12
Enter nay number :10
22

Similarly, we can convert to float in a similar way;

```
a = float(input("Enter any number :"))
b = float(input("Enter nay number :"))
c = a + b
print(c)
```

Result will be in float even if I put integer values because it will first be converted to float:
Enter any number :1
Enter nay number :2
3.0

9. **Changing Case:**
   Like all the object string is also object on which we can apply different functions. We can change case of strings.
   - We can convert string to upper case by using syntax : **print(string name.upper())**
   - We can convert whole string in lower case by using syntax : **print(stringname.lower())**
   - Last is converting string into title in which first letter will capital and rest of them will be small by syntax : **print(stringname.title())**

     We use all of these in the situation when we have specified the case and there are chances that user will enter it in any other case so first it will be converted.

10. **Changing Case – In Practice:**

```
name = "usman"
print(name.upper())
print(name.title())
```

Result:
This will print name string in upper and title case which is in lower case:
USMAN
Usman
Similarly, we can convert upper or title to lower case.

```
string = input("Are you a boy:")
if string=="yes":
    print(string.upper())
elif string=="Yes":
    print(string.upper())
else:
    print("please writ all letters in upper case")
```

Result:
Are you a boy:yes
YES
Running second time:
Are you a boy:Yes
YES

11. **Dictionaries:**
    Like list and tuple python also provides us dictionary to store many data elements. But in dictionary there is slight difference which is : In dictionary there are two things about one element, one is **key** and second will be its **value. For example,** *Name* can be key and *Usman* will be its value or any name. There is a colon between key and its value in dictionary.
    Syntax:
    NameofDictionary = {key:value , key:value , key:value}

    Items of dictionary don't have any index we get them randomly when we apply for loop on them.

    Key can have any data type and value also can have any data type. Value can list/tuple too. Like if we want to give more then one name.

12. **Dictionary – In Practice:**

```
my_dictionary = { "Height" : 6,"Name":"Usman","age":18}
print(my_dictionary)
len(my_dictionary)
```

It will return:
{'Height': 6, 'Name': 'Usman', 'age': 18}

13. Accessing Info from Dictionaries:

To access the info, we have to give key and it will return its value. If you will give value of key it and try to print key it will give type error.

Syntax will be : print(nameofdictionary[“key name”])

It will get more clearer if you see this code:

```
my_dictionary = { "Height" : 6,"Name":"Usman","age":18}
print(my_dictionary["Name"])
print(my_dictionary["Height"])
print(my_dictionary["age"])
```

Result:

Usman

6

18

Results will come randomly because there is no index to be consider.

14. Adding keys to Dictionaries:

Like list we can also add elements in the dictionary. For that we have to write:

Nameofdictionary[“key name”] = “Value”

Code:

```
my_dictionary = { "Height" : 6,"Name":"Usman","age":18}
print(my_dictionary)
#we want to add new element which is weight.
my_dictionary["weight"] = 90
print(my_dictionary)
```

Result:

{'Height': 6, 'Name': 'Usman', 'age': 18}

{'Height': 6, 'Name': 'Usman', 'age': 18, 'weight': 90}

If the key already exists then its value will be overwritten.

15. Adding a New Key to An Existing Dictionary – In Practice:

Adding key is shown in previous topic. It does not matter that key is integer and its value is string we can still get in the same way.

```
mydict = {1:"Usman",2:"Saad”,3:”Huzaifa”,4:”Hassaan”}
print(mydict[1])
print(mydict[2])
print(mydict[3])
print(mydict[4])
```

Result will be:

Usman

Saad

Huzaifa
Hassaan

- We have a feature to delete the key and its value from the dictionary.
  Syntax: del name of dictionary[key name]
  Code:

```
mydict = {1:"Usman",2:"saad",3:"Huzaifa",4:"hassaan"}
print(mydict)
del mydict[1]
print(mydict)
```

  Result;
  {1: 'Usman', 2: 'saad', 3: 'Huzaifa', 4: 'hassaan'}
  {2: 'saad', 3: 'Huzaifa', 4: 'hassaan'}
- You can update value of key by just giving new value to the key.

```
mydict = {1:"Usman",2:"saad",3:"Huzaifa",4:"hassaan"}
print(mydict)
mydict[1] = "Usman Yaqoob"
print(mydict)
```

  Result:
  {1: 'Usman', 2: 'saad', 3: 'Huzaifa', 4: 'hassaan'}
  {1: 'Usman Yaqoob', 2: 'saad', 3: 'Huzaifa', 4: 'hassaan'}

- If you will give two values to a single key in single dictionary then the latest value at the end will appear only.

```
mydict = {1:"Usman",2:"saad",3:"Huzaifa",4:"hassaan",1:"Usman Yaqoob"}
print(mydict)
```

  Result:
  {1: 'Usman Yaqoob', 2: 'saad', 3: 'Huzaifa', 4: 'hassaan'}

- We can also check that if any key is in the dictionary or not. For that:
  We will use in key work as we used in list:

```
mydict = {1:"Usman",2:"saad",3:"Huzaifa",4:"hassaan",1:"Usman Yaqoob"}
print(3 in mydict)
```

  So, it will return:

  True

16. **Removing Info from Dictionary:**
    Done

17. Iterating over info from Dictionaries:

Python provides three ways to access different things from dictionary. We can get key and values separately and together too.

- First is to get Key we have to use function of **".keys()"** with the dictionary name.

```
mydict = {1:"Usman",2:"saad",3:"Huzaifa",4:"hassaan",1:"Usman Yaqoob"}
for key in mydict.keys():
    print(key)
```

It will return all keys in return:
1
2
3
4

- Secondly to get values we have to use **".values()"** with dictionary name.

```
mydict = {1:"Usman",2:"saad",3:"Huzaifa",4:"Hassaan",1:"Usman Yaqoob"}
for values in mydict.values():
    print(values)
```

It will return all the values;
Usman Yaqoob
saad
Huzaifa
Hassaan

- Last is, we can get both of them together by using with function **".items"** with the dictionary name.

```
mydict = {1:"Usman",2:"saad",3:"Huzaifa",4:"hassaan",1:"Usman Yaqoob"}
for items in mydict.items():
    print(items)
```

It will return both of them together;
(1, 'Usman Yaqoob')
(2, 'saad')
(3, 'Huzaifa')
(4, 'hassaan')

18. What can you store in Dictionaries?

We can store a value in a key which is normal. We can also store **variable, list, tuple and another dictionary in the key or in the dictionary.**

```
a = "usman"
usman = {1:a,2:123,3:"Hey"}
print(usman)
```

We just store string in a variable and gave that variable in the dictionary. Result will be:
{1: 'usman', 2: 123, 3: 'Hey'}

19. List of Dictionaries:

We can make a list and we can include **dictionary** in it as members.

```
hey = [1,2,{3:"Usman",4:"Ali"}]
print(hey)
```

It will return the list:

[1, 2, {3: 'Usman', 4: 'Ali'}]

20. Accessing info from a List of Dictionaries:

In this case getting list from dictionary is little different from accessing usual items. We will use to index of to reach the dictionary and if we want to access anything from the dictionary, we have to further give key.

```
hey = [1,2,{3:"Usman",4:"Ali"}]
print(hey[2])
```

This will actually return the list:
{3: 'Usman', 4: 'Ali'}

Now to access value from the dictionary you have to give key in square brackets in front of index.

```
hey = [1,2,{3:"Usman",4:"Ali"}]
print(hey[2][4])
```

Result:
Ali

21. Dictionary of Lists:

Here we will make a dictionary in which at least one key will have more than one value/list.

To access the element from the list we have to do opposite thing we have to first give key of list then further we will give index.

```
hey = {1:"Usman" , 2:"Saad" , 3:"Ali" , 4:["hauzaifa", "Daniyal"]}
print(hey)
print(hey[4][0])
```

It will first print the dictionary and then will give item that have 0 index which is **Huzaifa. Result will be:**
{1: 'Usman', 2: 'Saad', 3: 'Ali', 4: ['hauzaifa', 'Daniyal']}
Hauzaifa

22. Dictionary of Dictionaries:

It means we have a dictionary given to a key in the dictionary or we can say dictionary in dictionary. We will access the items int the way that in the both square brackets we will give the key we want.

```
hey = {1:"Usman" , 2:"Saad" , 3:"Ali" , 4:{1:"hauzaifa", 2:"Daniyal",3:"Hassaan"}}
print(hey)
print(hey[4][3])
```

In this we the first 4 in the brackets will approach the dictionary and the other 3 will print Hassaan.

Result:

{1: 'Usman', 2: 'Saad', 3: 'Ali', 4: {1: 'hauzaifa', 2: 'Daniyal'}}

hauzaifa