Session - 8

Match statement -

Syntax -

```
match (expression):
    case constant:
        - - -
        - - -
    case constant:
        - - -
        - - -
```

No use of break.

- → temporary variable (default in c)

Ex -

```
user_in = int(input("Enter a number"))
match user_in:
    case 1:
        print("1")
    case 2:
        print("2")
    case 3:
        print("3")
print("completed").
```

o/p ⇒ Enter a number 3
3
completed.

```
user_in = int(input("enter a number:"))
match user_in:
    case 1:
        print("1")
    case 2:
        print("2")
    case 3:
        print("3")
    case _:
        print("something else")
Print("Completed").
```

o/p ⇒ enter a number - 12
something else completed.

→ "_" default case is executed only when the above case statements are false.
(underscore)

→ match statements help us to handle different cases and is very similar to if-else statement.

Nested-for loop: for within a for loop

```
for ---:
    for ---:
```

Ex- 2D list

```
list_2d = [[1,2,3],[4,5,6],[7,8,9]]
for ele in list_2d:
    print(ele)
```

o/p ⇒ [1,2,3] [4,5,6] [7,8,9]

```
for ele_1 in ele:
    print(ele_1)
```

o/p ⇒ 1 2 3 | 4 5 6 | 7 8 9

Tuple Data type:

- represented with ( )
- It is a ordered datatype
- supports indexing and slicing

Ex-
```
a = (10,20,30)
print(a[0])
print(a[1])
print(a[2])
```
o/p ⇒ 10 20 30

- Tuple is an immutable datatype.

Ex- a = (10,20,30)
insert value 40 at the end of array.

```
a = (10,20,30)
a = list(a) # typecasting
a.insert(-1,40) X | a.append(40)
a = tuple(a)
print(a)
```
o/p ⇒ (10,20,40,30) | o/p ⇒ (10,20,30,40)

- we can omit the use of ( ).
( ) are not necessary.

Ex-
```
a = 10,20,30
print(type(a))
```
o/p ⇒ (10,20,30) <class 'tuple'>

```
a = 10,20,30 # treats as a tuple
⇒ a,b = 10,20 # tuple unpacking
print(a)
print(b)
```
o/p ⇒ 10 20

- a,b = 10,20,30 # throws an error.
- No. of variables and no. of values should be same with tuple unpacking.

Enumerate( ) -

Ex-
```
a = (10,20,30)
for idx,ele in enumerate(a):
    print(idx,ele)
```
o/p ⇒ 0 10 | 1 20 | 2 30

count and index operations in list -

```
a = (10,20,30,40,30,50)
print(a.count(30))
```
o/p ⇒ 2

```
a = (10,20,30,40,30,50)
print(a.index(30))
```
o/p ⇒ 2 by default it gives the index of first occurence.

```
a = (10,20,30,40,30,50)
print(a.index(30,3))
```
→ used as the start index to check the given element

o/p ⇒ 4

Ex-
```
fruits = ["apple","cherry","banana","banana","apple","cherry"]
fruit_to_search = "cherry"
idx = 0
for idx,fruit in enumerate(fruits):
    if fruit == fruit_to_search:
        print(fruits.index(fruit_to_search,idx))
        print(idx)
```
o/p ⇒ 1 5

- To know the operations that can be performed on particular datatype, we can use directory function.

Ex- print (dir(tuple))

o/p ⇒ add, class, eq, ge, new, reduce, str, count, index - - - - etc.

Ex- print(dir(list))

o/p → append, clear, copy, count, extend, index, insert, pop, remove, reverse, sort, etc.

- Tuple doesnot have sort operation but it performs sorted operation

Ex-
```
a = (4,2,3,1)
a = sorted(a)
print(a)
```
o/p ⇒ [1,2,3,4]
it results a list

```
a = (4,3,2,1)
a = tuple(sorted(a))
print(a)
```
o/p ⇒ (1,2,3,4)

Count number of pairs that are equal to a given sum

Given list → [-1, 1, 5, 7, 100]

target sum → 6

-1 7
1 5
5 1

no. of pair → ?

```
given_list = [-1,1,5,7,100]
target_sum = 6
num_pairs = 0
for ele in given_list[:-1]:
    for ele1 in given_list[1:]:
        if ele + ele1 == target_sum:
            num_pair += 1
            print(ele,ele1)
    print(num_pairs)
```

o/p ⇒ 3