17-02-2025

Training Day - 13

PROGRAM ON STRING OPERATION AND LIST METHODS

……….New Program……

```
# L=[] #
e=10
# L.append(e) #
print(L)
# e=20
# L.append(e) #
print(L)
```

……….*New Program*……

```
# s="cetpa"  #
r=s.upper() #
print(r)
```

……….*New Program*……

```
# L1=[10,20,30]
# L2=L1.append(40) #
print(L2, L1)


# # #New Program  #
s="cetpa infotech"
# r=s.replace("e","*") #
print(r)
```

LIST METHODS

……….*New Program*……

```
# L=[10,20,30]
# print(id(L))
# L.append(40) #
print(L)
# print(id(L))
```

……….*New Program*……

```
# L=[10,20,30]
# print(id(L)) #
L[0]=100
# print(L)
# print(id(L))

# #New Programm #
L=[10,20,30]
# L.extend(40)              #Error: Extend method works on iterator
# print(L)

# #New Program #
L=[10,20,30]
# r=L.pop(1)             #index is optional to pass
# print(L) #
print(r)

# #New Program
# L=[10,20,30,40,50,60] # i=2
# L.pop(i) #
print(L)

# #New Program
# L=[10,20,30,20,40]
# ele=20
# r=L.count(ele) #
print(r)
# print(L)

# #New Program #
L=[10,20,30]
# print(id(L))  #
L1=L.copy() #
print(id(L1)) #
print(L,L1)
```

18-02-2025

Training Day - 14

Dictionary:

1. Dictionary is a collection of heterogeneous data types.
2. Dictionary is a collection of key-value pairs. One key-value pair is called one item.
3. Dictionary is mutable in nature
4. Dictionary can have only unique keys ie can't have duplicate keys.
5. Dictionary is a collection of unordered items. Dictionary elements don't have direct index.

Syntax:

{comma separated key-value pairs}
dict_var={key1:value1,key2:value2,key3:value3 .... }

Syntax to access elements of a dictionary:

dict_var[key]

```
# #New Program
# d={1:10,2:50,"CETPA":80,90:"ABC",50:[2,3,4]}
# print(d[1])
# print(d["CETPA"]) #
print(d[50])

# #New Program: Aditi says if duplicate keys are there #
d={1:10,2:20,3:30,2:50,4:40,2:80}
# print(d)

# #New Program
# d={1:10,2:20,3:30}              #d address 1000
# print(d,id(d))
# d[2]=80                    #d address 1000
# print(d,id(d))
```

Benefit of using dictionary:

In real life, in almost all cases, we are not aware about the index of a data rather we are aware about the actual data elements. Now if data is stored in a list or tuple, and if we want to find the index of a particular element and in case the element is far away from starting or end point, then it takes a lot of time to search the element in a big data. But the better approach can be, we can store the data in a dictionary and can make the unique values of data as a key like customer id, employee id, student roll no etc. And now if we are aware about the key, we can immediately access the data element.

```
cus_dict={10:["Vikas",39,9212468020],20:["Anil",41,9654444252],................ }
id=20 print(cus_dict[id])
```

In dictionary, the keys are first converted to hash codes,

```
    List
    L=[10,20,30,40]
    """

    # L=[10,20,30,40,50,60,7
```

19-02-2025

Training Day - 15

CATEGORIES OF FUNCTIONS:

1. Required Argument Functions: Where the no of arguments and sequence of arguments should match in formal and actual. Till now all functions we created, were required argument type
2. Keyword Argument Functions: Where the no of arguments should be same in formal and actual but there position can vary.
3. Default Argument Functions: Where we assign a default value to a variable. Now while calling, if we pass the value in actual then new values is used otherwise default value is used.
4. Variable Length Argument Functions (Tuple Based): In formal parameter, we add one extra star immediately before variable name. Python recommends that the formal variable name should be 'args' in case of variable length argument functions
5. Variable Length Keyword Argument Functions (Dictionary Based): In formal parameter, we add two extra stars immediately before variable name. Python recommends that the formal variable name should be 'kwargs' in case of variable length keyword argument functions
6. Lambda Functions: Anonymous Functions: Lambda functions are single liner anonymous functions. lambda arguments_passed:expression_calculated&returned

""""

1. Formal Arguments

```
# #New Program
# def add(a,b):
#     return a+b
#
# u,v,w=5,7,9
# s1=add(u,v,w)       #Error
# print(s1)
```

2. Keyword Arguments #addCustomer(id=id,name=name...)

```
# def sub(a,b):
#     return a-b
# u,v=5,7       #u-v
# r=sub(b=v,a=u)
# print(r)
```

3. Default Arguments

```
# def add(a=1,b=2):
#     return a+b
# r1=add()       #r1=3
# r2=add(5)      #r2=7
# r3=add(b=9)    #r3=10
# r4=add(5,7) #r4=12
# r5=add(b=8,a=2) #r5=10
# print(r1,r2,r3,r4,r5)
```

4. Variable Length Argument Functions (Tuple Based):

```
# #New Program
# def func1(*t):       #Formal Variable name is t
#     print(t)
# func1(2,3,4)
# func1(10,20,30,40,50,60)
# func1()
```

FUNCTION POINTER: Function pointer is a variable which holds the address of a function.

```
def func1():
    pass
a=func1()
```

here a is a function pointer which is holding the address of a function.

lambda arguments_passed:expression_calculated&returned

```
# print(lambda a,b:a+b)

# #New Program
# a=lambda a,b:a+b         #a is a funciton pointer
# r=a(5,7)
# print(r)

#New Program
add=lambda a,b:a+b         #a is a funciton pointer
r=add(5,7)
print(r)
```

20-02-2025 **Training Day – 16**

OOPS: Object Oriented Programming: In python we have concept of classes and OOPS base is Class

Class: Is a collection of variables and functions. In OOPS or in class, generally the functions are called methods.

Python Supports Modular Approach Of Programming: We can develop any big project without OOPS also but complexity of the project will be increased a lot. Using OOPS, we can decrease the complexity of the project, the project is made scalable using OOPS, we can easily modify and enhance the project later using OOPS.

Syntax to create a class:

```
class Class_Name:
   variables and methods ie block of code
.
```

How to create object of any class:

```
obj_name=class_name(arguments)
obj_name=class_name()
```

```
# #New Program
# class C1:
#     pass
# obj=C1()         #obj is object of C1 Class
# print(type(obj))
# print(obj)
```

If we are calling a method which is created inside a class, then while calling from actual parameters, first object is passed to formal parameters and then rest of the arguments are passed.
"""

```
# #New Program
# d1={1:10,2:20,3:30}       #d1 is of dict type
# d2={11:100,12:200,13:300}
# res=d1.values()      #If values is a function values(d1)
# print(res)
```

When we call a method which is inside a class then from formal parameters object is passed to the first argument of actual parameters. Python recommends that the object name in formal parameters should be self.

```
# #New Program
# L1=[1,2,3]
# L1.append(5)
```

Class represents real time entity.

One of the benefits of using OOPS is that we can easily represent real time entities using classes

Syntax to call a method of a class:
obj_name.method_name(arguments)

How to access an instance variable:
obj_name.variable_name

```
# #New Program
# class C1:
#     pass
# obj1=C1()
# obj1.a=5     #a is a variable of obj1 object
# print(obj1.a)
# obj1.b=7
# obj1.c=9
# print(obj1.a,obj1.b,obj1.c)
# obj2=C1()
# obj2.a=20
# obj2.b=30
# print(obj1.a,obj1.b,obj1.c,obj2.a,obj2.b)
```

21-02-2025

Training Day – 17

Create A Generalized Function which takes any no of arguments and return the multiplication of all arguments using variable length keyword argument function.

```
# #BLL
# def mul(*args):      #args=(2, 3, 4, 5, 6)
#     r=1
#     for i in range(len(args)):
#         r=r*args[i]
#     return r
# #PL
# r1=mul(2,3,4,5,6)
# r2=mul(1,2)
# print(r1,r2)
```

""""
Create your own generalized index function. Print all the matching index positions of an element present in a list [2,3,4,5,6,2,3,4,2,3,4,2]. Take the element as input from the user.
""""

```
# #New Program
# L=[2,3,4,5,6,2,3,4,2,3,4,2]
# ele=2
# for i in range(len(L)):      #i=0,1,2,...n-1
#     if(L[i]==ele):
#         print(i)
```

Constructor: is a method, which is called automatically everytime we create an object in python. In Python the name of the constructor is fixed ie __init_()

""""

```
# #New Program
# class C1:
#     def __init__(self):       #Constructor
#         print("CETPA")
#
#  ob1=C1()
#  ob2=C1()
# ob3=C1()
```

Generally in programming in real world, the variables of all objects of a class are common like all customers will have same variables like id, name, age, mob so we mostly create the variables inside constructor.

```
# class Customer:
#     def __init__(self):  #self=1000, self=2000
#         self.id=0        #1000.id=0, 2000.id=0
#         self.name=0      #1000.name=0
#         self.age=0       #1000.age=0
#         self.mob=0       #1000.mob=0
# cus1=Customer()       #cus1 1000, self 1000
# print(cus1.id,cus1.name,cus1.age,cus1.mob)
# cus2=Customer()      #cus2 2000 , self 2000
# print(cus2.id,cus2.name,cus2.age,cus2.mob)
```

Now class or static variables and methods. These variables or method are like normal variables or functions which we have studied outside class.

How To Create Static Variables:

Same syntax like outside class. Directly inside class, assign
the value
var_name=value

How To Access Static Variables: using class name
class_name.var_name
Static variables will be common variables