8-10-2024

Training Day - 16

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)
```