Date 18/12/21

CHAPTER → 9

User Defined Functions

Introduction → फंक्शन में built-in library function के अलावा user स्वयं फंक्शन बना सकता है। यूजर के द्वारा बनाए गए फंक्शन user defined function कहलाते हैं।

एक बार फंक्शन बनाने के बाद उसे बार-बार कॉल करके प्रयोग किया जा सकता है।
फंक्शन बनाने तथा प्रयोग करने के लिए दो स्टेप हैं -

1. Define a function
2. Calling to function or Invoking a function

(1.) Define a function → Def keyword का प्रयोग करके नया फंक्शन बनाया जाता है तथा उसमें Codes लिखे जाते हैं। यह फंक्शन की definition होती है। फंक्शन कोई value return कर भी सकता है। और नहीं भी।

Syntax:
```
def <function name>(<list of argument>)
    <Statement>
    return <expression>
```

Example:
```
def sum(a,b)
    t = a+b
    return t
```

(2.) Invoking a function → फंक्शन define करने के बाद इसके result को प्राप्त करने के लिए फंक्शन को कॉल की जाती है। फंक्शन कॉल कहलाती है।
फंक्शन के नाम invoking का प्रयोग करके कॉल की जा सकती है।

Syntax: <function name>(<list of argument>)

Example:

```
a = sum(5,2)
print a
print(sum(5+a))
```

→ 7
→ 12

Function Argument :-

फंक्शन के हैडर मे प्रयोग किये गये variables function argument कहलाते हैं।

ये तथा calling function से आने वाली value को प्राप्त करते हैं स्टोर करते हैं।

पायथन तीन प्रकार के argument support करता है -

(i) Positional argument
(ii) Default argument
(iii) Named argument

(i) **Positional argument** → जिन arguments को उनकी संख्या के समान मानो की आवश्यकता होती है. positional argument कहलाते हैं।

```
def sum(a,b)
    t = a+b
    return t
x=y=z=6
print(sum(5,6))
c = sum(x,y)
d = sum(5,z)
```

(ii) Default argument → फंक्शन हेडर में जिन arguments को पहले से कुछ मान दिये जाते हैं वह default argument होते हैं।

```
def sum (a,b=4)
    t = a+b
    return t
print (sum (5,6) → 11
c = sum (x,y) → 10 , d = sum (5,z) → 11
e = sum (9) → 6
```

x=y=z=6

(iii) Named / Keyword argument → फंक्शन में argument को भेजी गई value पर पूरा कन्ट्रोल रखने के लिए named argument प्रयोग किये जाते हैं। इसमें calling के समय sequence में मान भेजने की आवश्यकता नहीं होती।

```
def sum (b=4, a=5, c=6)
    t = a+b+c
    return t
z = sum (4,5,6)
x = sum (c=4, a=5, b=9)
y = sum ( )
```

Type of Function :-

Function argument तथा return value के आधार पर फंक्शन चार प्रकार के होते हैं -

1. Function with input arg. and return value
2. Function with input arg. and without return value
3. Function without input arg. and with return value
4. Function " " " and return value

1. **Function with Input and Output :—**

इस प्रकार के फंक्शन एक या अधिक value argument के रूप में इनपुट करते है तथा एक value return करते हैं।

Program: एक प्रोग्राम लिखें जिसमें एक नंबर इनपुट कर उसके Factorial की गणना करें।

```
def Fact (a):
    f = 1
    For i in range (a, 0, -1):
        f = f x a
    return f
# main program
n = int (input (" Enter any no."))
print (fact (n))
```

2. **Function with input & without return :**

इस प्रकार के फंक्शन एक या अधिक argument इनपुट कराते है तथा कोई भी value return नही करते।

Program: एक प्रोग्राम लिखें जिसमें एक नम्बर इनपुट कर उसकी टेबल को 10 तक प्रिंट करे।

```
def table(a):
    For i in range (1, 11):
        print (a, "*", i, "=", a * i)
# main program
n = int (input (" enter any value")
table (n)
```

3. Function without input arg. & with return :-

इस प्रकार के फंक्शन में input argument नहीं होते परन्तु यह एक value return करते हैं।

Program: पहले 50 सम संख्याओं की गणना कर प्रिंट करने के लिए एक प्रोग्राम लिखे।

```
def even_sum():
    t=0
    for a in range(1,51):
        t=t+a*2
    return t

print(even_sum())
```

4. Function without input & output :-

इस प्रकार के फंक्शन ना तो कोई argument उपयुक्त करते हैं और ना ही कोई value return करते हैं।

Program: निम्न पैटर्न उत्पन्न करने के लिए एक प्रोग्राम लिखे।

```
def pattern():
    for a in range(1,6):
        for b in range(a,5):
            print(" ", end=" ")
        for c in range(1,a+1):
            print(c, end="")
        print(" ")
# main program
pattern()
```

```
    1
   12
  123
 1234
12345
```

Recursion —

जब एक फंक्शन स्वयं को call करता है तब यह प्रक्रिया calling Recursion कहलाती है।

Recursion एक लूप के समान कार्य करता है जिसे रोकने के लिए एक terminate condition दी जाती है।

```
def test():
    test()
```

```
def test():
    retest()
def retest():
    test()
```

Program: पहले प्राकृतिक संख्याओं का योग प्रिन्ट करने के लिए N रिकर्शन प्रोग्राम बनाएं।

```
def sum(a):
    if a==1:
        return 1
    else:
        return a + sum(a-1)
# main program
n = int(input("enter any value"))
print(sum(n))
```