Python mein function kya hota hai?

Python mein function ek aisa block of code hai jo kuch specific tasks perform karta hai. Functions ko ek baar define karne ke baad, aap unhe multiple baar call kar sakte hain, jo code ko clean aur reusable banaata hai.

- Example and Syntax of a function.

```
def func1():
    print("jay shree ram")
```

- Fucntion Call. Function ko call karte waqt uska naam likhte hain, aur agar koi argument hai toh woh bhi pass karte hain.

```
func1()
```

function ke types:

- Built-in Functions: Ye functions Python me already defined hote hain. Example: print(), len(), sum(), etc.
- User-defined Functions: Ye functions hum khud banate hain using the def keyword.
- Example:

```
def greet():
    print("Namaste")

# call krenge
greet()
```

Function with argument.

- Python mein function with arguments ka concept simple hai. Aap function banate ho aur usmein values (arguments) pass karte ho. Yahan ek example de raha hoon:
- Example:

```
def greet(name):
    print(f"राम राम, {name}!")

greet("Loha don")  # "Loha don" yeh argument function ko pass ho raha hai
#Output राम राम, Loha don!
```

- Is example mein greet ek function hai jo ek argument name ko accept karta hai. Jab hum greet("Loha don") call karte hain, toh function ko "Loha don" argument milta hai, aur output hota hai:

Function with parameter.

- Python mein parameter wo variable hote hain jo function ke definition mein diye jate hain. Jab function ko call karte hain, toh unhe arguments ke through values pass ki jati hain. Toh, parameter aur argument ka difference yeh hai:
- Example:

```
def greet(name, age):
    print(f"राम राम, {name}! You are {age} years old.")

# Function call with arguments
greet("Loha don", 25)

#Output राम राम, Loha don! You are 25 years old.
```

- name aur age function ke parameters hain.
- Jab hum greet("Loha don", 25) call karte hain toh "Loha don" aur 25 arguments hain.
- Parameter: Function ke definition mein jo variable likhe hote hain, unhe parameter kehte hain.
- Argument: Function ko call karte waqt jo values diye jate hain, unhe argument kehte hain.

Function with default parameter.

- Python mein default parameters ka matlab hota hai ki function ke parameters ko default value assign kar di jati hai. Agar function ko call karte waqt wo parameter provide nahi kiya jata, toh default value automatically use hoti hai. Yeh concept function ko zyada flexible banata hai.
- Agar function ko call karte waqt kisi parameter ki value nahi di gayi, toh wo default value use karega.
- Example:

```
def greet(name, age=25):
    print(f"राम राम, {name}! apki  {age}")
    print(f"राम राम, {name}! apki  {100}")
```

- name parameter ko zaroori value deni hoti hai jab function call karein.
- age parameter ko default value 25 di gayi hai, agar hum age pass nahi karte toh wo 25 lelega.

Function with return statement.

- Python mein return statement ka use function se koi value wapas bhejne ke liye kiya jata hai. Jab aap return statement likhte ho, toh function execution wahan se ruk jata hai aur jo value return ke baad likhi hoti hai, wo function ko call karne wale jagah pe wapas chali jaati hai.
- ExampleL

```
def add(a, b):
    result = a + b
    return result
    # Yeh return statement result ko wapas bhej raha hai

sum_result = add(5, 7)
# add function ko call kiya jaa raha hai
print(sum_result)  # Output: 12
```

- add function mein return result likha gaya hai, jo a + b ka result return karta hai.
- Jab hum add(5, 7) call karte hain, toh result ki value 12 wapas milti hai, jo sum_result variable mein store hoti hai, aur phir hum use print karte hain

Recursion in Function.

- Recursion ek aisa concept hai jisme ek function apne aapko call karta hai. Recursion tab useful hoti hai jab hum kisi problem ko chhote sub-problems mein todna chahte hain, jo ki ek hi structure mein solve kiye ja sakte hain.

Recursion ka Basic Concept:

- Base case: Jab recursion ko rukna hota hai. Agar base case nahi diya gaya toh recursion infinite loop mein chali jayegi.
- Recursive case: Jo part function ko apne aapko call karne ke liye banaata hai.

Example: Sabse common example factorial calculation ka hai. Factorial of a number n is the product of all positive integers less than or equal to n.

```
def factorial(n):
    if (n == 1 or n == 0): # Base case
        return 1
    else:
        return n * factorial(n - 1) # Recursive case

n = int(input("Enter a number: "))
print(f"The factorial of this number is: {factorial(n)}")
```

- Base case: if n == 1, n == 0 jab n 1 ho or 0 toh function 1 return karega. Yeh recursion ko rokta hai.
- Recursive case: n * factorial(n - 1) yeh line function ko apne aapko call karti hai, jab tak n 1 nahi ho jata.

Recursion ka Flow:

- Agar hum factorial(5) call karte hain, toh recursion ka flow kuch is tarah hota hai:

```
                 factorial(0) = 1
                 factorial(1) = 1
                 factorial(2) = 2 * 1 = 2
                 factorial(3) = 3 * 2 = 6
                 factorial(4) = 4 * 6 = 24
                 factorial(5) = 5 * 24 = 120

                 # Final output: 120
```

Fayda kya h Recursion ka.

- Recursion ka advantage yeh hai ki complex problems ko clean aur simple code ke saath solve kiya ja sakta hai. Lekin dhyan rahe ki recursion ka depth zyada na ho, kyunki bahut deep recursion se stack overflow ho sakta hai.

- recursion ye tab jada use hogi jab aap advance level pr code likhoge. aur shurat mein ye bilkul samaj ni ayga bar bar practice krni pdegi

- recursion ek trha se loop hi hota h jo kaam loop mein kiye jata h vhi same recusion mein kre jaa skte h bs recusion mein task perform krne se code ki line kam hoti kam line mein hi kaam ho jata h example dekho upr factorial nikala hmne recursion ki madad se ab loop ki madad se krte h dekho niche

```python
n = int(input("Enter number: "))
result = 1
for i in range(1, n + 1):
    result *= i

print(f"Factorial of {n} is {result}")
```

- range() ye ek in-built function hota h.
- range() function mein 3 parameters hote hain:
- range(start, stop, step)
- start: Ye wo number hai jahan se range shuru hoga. (optional)
- stop: Ye wo number hai jahan range khatam hoga. (required)
- step: Ye wo value hai jo har iteration mein increment hota hai. (optional)

Recursion aur loop mein kya best h?

- Recursive functions zyada memory consume karte hain (stack frame banane ke kaaran). Isi wajah se recursion slow ho sakta hai, aur stack overflow ka risk hota hai.

- Loops generally faster aur memory-efficient hote hain, kyunki ye extra stack frames create nahi karte.

- Mein tho ye khaunga starting mein ise focus mt krio bs ise sikh le ki agee agr iska use aya tho tuje ek dam se smj aa jaye dekh kar ye jadar tar bade project mein kaam ayga.

Lambda function

- Lambda function ek aise function hota hai jo bina name ke define hota hai. Iska use simple functions ko ek line mein define karne ke liye kiya jaata hai. Lambda function ko anonymous function bhi kaha jaata hai.
- Lambda function ka syntax kuch is tarah hota hai:

```
lambda arguments: expression
```

- lambda keyword hai.
- arguments wo values hain jo function ko input diye jaate hain.
- expression wo calculation hai jo function perform karega aur result return karega.

- Example 1: jese muje ek lambda function bnana h jo do value ko multiply kre yani x*y.

```
number = lambda x,y: x*y
multiply = number(2,5)
print(multiply) #Output = 10
```

- Example 2: cube nikalo lambda function ka use krke

```
num = int(input("enter number: "))
number = lambda num: num ** 3
cube = number(num)
print(cube)
```

hme lambda function use krna h ya normal def vala?

- Lambda function: dekho lambda function chote task perform krne k liye use hota h jese mene example diye h upr jo ek line mein ho perform ho jate h.

- Def function: lekin complex task perform krne k liye aur jisme multiple steps ya conditions hain, tab hum def ka use krenge.

fucntion with *args

- *args Python me ek tarika hai jo hume function ke andar variable number of arguments pass karne ki flexibility deta hai. Agar hume nahi pata ki function me kitne arguments pass honge, to *args ka use karte hain.

Detail Explanation:

- *args ek special syntax hai jo function mein multiple arguments accept karne ke liye use hoti hai.

- args ek convention hai, aap isko kisi aur naam se bhi likh sakte ho (e.g., *numbers, *values).
- Ye ek tuple banata hai, toh aap iske elements ko loop ya indexing ke through access kar sakte ho.
- Sytax

```
 def function_name(*args):
# Function Body
```

- Example:

```
def add_numbers(*args):
    result = 0
    for num in args:
        result += num
    return result

# Function call
add_numbers()
sum_of_two_number = add_numbers(10, 20)
print(sum_of_two_number)         # Output: 30

sum_of_five_numbers = add_numbers(1, 2, 3, 4, 5)
print(sum_of_five_numbers) # Output: 15
```

Explanation:

- add_numbers function ko kitne bhi arguments pass kar sakte ho.
- Saare arguments args tuple ke andar store ho jaate hain.
- Loop ke through hum unka sum nikalte hain.

Mixed Arguments with *args:

- Agar aapke function mein fixed arguments bhi hain aur *args bhi, toh fixed arguments pehle define hote hain.

```
def greet_message(greeting, *names):
        for name in names:
            print(f"{greeting}, {name}!")

# Function call
greet_message("राम राम", "Loha", "Shyam", "Mohan")

#Output
राम राम, Loha!
राम राम, Shyam!
राम राम, Mohan!
```

Important Points:

- *args sirf positional arguments ko handle karta hai.
- Agar keyword arguments (key-value pairs) ka use karna ho, toh **kwargs ka use karte hain.

Function with **kwargs:

- **kwargs ka use Python me tab hota hai jab hume kisi function me variable number of keyword arguments pass karne hote hain. Yeh ek dictionary ki tarah kaam karta hai, jisme arguments ke keys aur unki values store hoti hain.
- Syntax me **kwargs ka matlab hai "keyword arguments." Iska naam kwargs ke alawa kuch aur bhi ho sakta hai, par ** lagana zaroori hai.
- Example:

```
def student_details(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

# Function call
student_details(name="Loha", age=20, grade="A")

# Output
name: Loha
age: 20
grade: A
```

Points to Remember:

- **kwargs dictionary me data store karta hai.
- Iska use tab hota hai jab hume pata nahi hota ki function me kitne keyword arguments pass honge.
- Yeh *args se alag hai, jo tuple me non-keyword arguments store karta hai