* Functions :-

A function is a block of reusable code that performs a specific task.
- instead of writing the same code again and again, we can put that code inside a function and use it whenever we need by just calling the function.

Reusability - you can write the code once and use it multiple times.

organization - they devided a big program into smaller, manageable parts.

Readability - The program becomes easier to read and understand.

Maintainability - if you need to take make a change, you do it in one place (inside func), not every where in the program.

Avoiding Repetion - Reduces writing the same lines of code again and again

With function:-

```
def greet():
    print ("Hello, welcome to python!")
```

o/p:- Hello, welcome to python.

* the message is written only once inside the function
* you can reuse it anywhere by calling "fun: name".

Without function:-

```
# printing message many times (repeated code).
print ("Hello, welcome")
print ("Hello, welcome")
```

* Here the same line of code is Repeated three times
If you want change the mesg, you must edit it in every line.

Types of functions :-

1. Built-in function.
2. User defined function → 1. function without i/p and without return
3. Lambda function. 2. function with i/p and without return
3. function without i/p and with return
4. function with i/p and with return.

1) Built in function :-

Built in functions are the functions that are already available in python.
you dont have to create them - you can use them directly in your pr

print (), type (), len(), input (), max (), min (), sum (), abs()
round (), range ().

ex:- * name = input ("enter your name :")
print ("Hello :", name)

o/p :- Enter your name :
Nagi
Hello Nagi

* text = "python"
print ("length of text :", len (text))
print ("type of Name variable :", type(name)).

o/p :- L of T :6
type of name variable
<class 'str'>.

2) user - defined function :-

this function is created by the user using the 'def' key word to perform a specific task. (you can create your own function).

syntax :-

```
def fun_name (parameters):
    statements.
    return value.
```

def → key word used to define (create) a new function.
fun_name → name you give to your function.
parameter → optional value passed into the fun,
statement (s) → the block of code that performs the task.
return value → send a value back to where the function was called optio

(i) function without input and without return :-

* The function does not take any input (no values are passed inside the parantheses when calling it).
* The function does not return anything back to the main program.
* It just perform some action → ex:- printing o/p or showing a message

Syntax:-

```
def fun_name():
    statement(s)

# call
fun_name()
```

ex:- 1) printing a message.

```
def greet():
    print("Hello! welcome to python programming.")

# call
greet()
```

o/p :- Hello! welcome to python progr

Explain:- The function greet() does not take input
It simply prints a message.
It does not return anything.

② check even or odd :-

```
def even_odd():
    num = int(input("enter a num:"))
    if num % 2 == 0:
        print("even number")
    else:
        print("odd number")

# call
even_odd().
```

o/p :-
enter a num - 6
even num

Expl:- * No input is passed when calling (even_odd())
* It ask for input inside the function
* It prints whether the num is even or odd
* It does not return anything.

* function without I/p and without return → No input and No return → It prints a message or result directly.

Ex:- simple interest :-

```
def si():
    P = float(input("enter a principle amount:"))
    t = float(input("enter a time:"))
    r = float(input("enter a rate:"))
```

```
    si = (p*t*r)/100
    print ("the simple interest is:", si)

# call
si()
```

O/P:-
enter principal amount: 5000
enter a time: 1
enter a rate: 2
The simple interest is: 1000

2) function with input and without Return:-

* The function with input and without return means.
* The function takes input value (parameters) when it is called.
* But it does not return any value to the main program.
* It usually performs an action, like printing or displaying a result.

Syntax:-

```
def fun_name (parameter1, par2, ...):
    statement
#call
fun_name (arg1, arg2, ....)
```

Explanation:-

1. def :- used to define the function
2. fun_name :- name of the function
3. inside () - we write parameters, which are inputs for the function.
4. statements - the code that performs a specific task.
5. No return stmt - means the fun just executes and prints, but doesn't send any result back.

Ex:- print sum of 2 num.

```
def add (a,b):          # function with i/p parameter.
    print ("sum =", a+b)
# function call with arg.
add (5,3)
```

O/P:- sum = 8

Explain:-
- a and b are i/p ~~arguments~~ parameter
- 5 and 3 are arguments passed during the function call.
- function print the result - but does not return it.

ex: check even or odd (with i/p)

```
def even_odd (num):
    if num%2 ==0:
        print (num, "is even").
    else:
        print (num, "is odd").
# function call.
even_odd (10)
even_odd (7).
```

o/p:- 10 is even
7 is odd.

Explain:-

- The function takes i/p (num)
- It checks if num is even or odd.
- It prints the result (no return value).

* function with i/p and without return → input → No Return → print result only.

ex:-
```
def add (a,b):
    print (a+b)
```

o/p:- The si is : 1000.

ex:-
```
def si2 (p,t,r):
    si = (p*t*r) /100
    print ("The si is :",si)
# call
si2(50000,1,2)
```

*
```
pa = float (input ("enter a principle amount :"))
ti = float (input ("enter a time"))
ra = float (input ("enter a rate :"))
```

o/p:- principle amount : 50000
time : 2
rate : 4.

3) function without input but with Return:-

* The function does not take any input values (no parameters).
* But it returns a value back to the main program using the return statement.

The fun does the work inside, and then sends the result back (return it) instead of printing directly.

Syntax:-

```
def fun_name ():
    statements
    return value.
#call
variable = fun_name ().
```

1. No input — parantheses are empty "()" so no data is passed.
2. calculation or operations happen inside the function.
3. return stm — sends the result back to where the function was called.
4. The returned value can be store in a variable or used in expression

Ex:- Return the sum of two numbers:-

```
def add():
    a = 10
    b = 5
    result = a + b
    return result      # return value.

# fun_call
sum_value = add()
print("sum =", sum_value)
```

op:- sum = 15

exp:- • function has no input (a and b are declared inside it).
• It return the sum using return result.
• The returned valued is stored in sum_value.

* Even or odd:-

```
def even_odd():
    num = int(input("enter a number:"))
    if num % 2 == 0:
        return "even"
    else:
        return "odd"

# call
result = even_odd()
print("The number is", result)
```

op:-
enter the num: 6
The num is even.

Explain:- • The function takes no input when called.
• It return either 'even' or 'odd'
• The returned value is stored in result and printed later.

* function without input but with return → No input → Return → Return values to program.

Ex:- def add(): return a+b.

ex:- def si3():

```
    p = float (input ("enter a principle amount :"))
    t = float (input ("enter a time"))
    r = float (input ("enter a rate"))
    si = (p*t*r)/100
    return si
```

# call
si3()

(or)

return si,p,t,r

o/p:-
enter a principal amount :50000
enter a time :2
enter a rate :3
3000.

o/p:- (3000, 50000, 2, 3)

ex:-
```
siv,pv,tv,rv = si3()
```

o/p:- enter a principal amount : 50000
enter a time : 2
enter a rate : 4.

* siv o/p - 4000
* pv o/p - 50000
* tv o/p - 2
* rv o/p - 4

4) function with i/p and with return:-

* the function takes i/p values (parameters) when it is called.
* It return a value back to the main program using the return statement.

(or)

the function accepts data, processes it and sends the result back to where it was called.

syntax:-

```
def fun_name (p1, p2 ... pn):
    statements
    return value.
# call
fun_name (p1, p2 ... pn).
```

explain:-

1. input parameters - The function recieves data when it's called.
   ex:- def add(a,b): → a and b are inputs.
2. process inside -
   The function perform calculation or logic using those inputs.

3. Return value:

~~The function performs calculation or logic using Those inputs.~~

It sends the result back using the return keyword.

4. function call:

The returned value is usually stored in a variable or printed.

Example:- Addition of 2 num.

```
def add (a,b):
    result = a + b
    return result
#call
sum_value = add (10,20)
print ("sum =", sum_value).
```

o/p:- sum.

explain:-

- a and b → input parameters
- function calculate sum.
- Return the result to the main program.
- stored in sum_value.

Ex:- check even or odd.

```
def even_odd (num):
    if num % 2 == 0:
        return "even"
    else:
        return "odd"
# call
result = even_odd (9)
print ("The num is", result).
```

o/p:- the num is odd

explain:-

- input → num = 9
- function checks and return "odd"
- The result is stored and printed later.

Ex:-

```
def sip (p,t,r)
    si = (p*t*r)/100
    return si
# call
sip (300000, 2, 3)
```

o/p:- 18000.

Create function name to print sum of given series range of values using with i/p and without return.

```
def sum_val (stv,spv):
    s=0
    for i in range (stv,spv+1,1):
        s=s+i
    print (f "The total sum from {stv} to {spv} is {s}")
```

o/p:- The total sum from 1 to 10 is 55

call
sum_val (1,10).

Create a function to print prime num from 1 to 20 using without input and without return.

```
stv = int (input ())
stp = int (input ())
for i in range (stv, stp+1, 1):
    for j in range (2,i,1):
        if (i%j ==0)
            break
    else:
        print (i).
```

o/p:- 2
10

2
3
5
7

(or)

```
def prime (stv,stp):
    pn = []
    for i in range (stv,stp+1,1):
        for j in range (2,i,1):
            if (i%j ==0):
                break.
        else:
            pn.append (i)
    return pn.
```

o/p:-
[2,3,5,7]

call
prime (2,10)

Create a function to find factorial of given num using with i/p and with return.

```
def fact (num):
    f=1
    for i in range (1,num+1,1):
        f=f*i
    return f.
```

o/p:- 3628800.

call
fact (10).

No	function type	input (pa)	Return value	Description	example	output
1.	without input & without return	NO Just do and show	NO	function perform task, doesn't take input or return value.	python def greet(): print("Hello!")	print → Hello!
2.	with input and without return	YES Take IP, show result	NO	Takes input, perform task, but doesn't return value	python def add(a,b): print(a+b)	print → sum = 8
3.	without input and with return	NO work inside, send back	YES	Doesn't take input, but returns a result	python def add(): return 5+3	Return → 8
4.	with input and with return	YES Take input, send back.	YES	Takes input, processes it, and return result.	python def add(a,b): return a+b.	Return → 8

* LAMBDA Function:-

A lambda function is an anonymous (nameless) function in python - That means it has no name like a normal function.

It is used to write small, simple function in a single line using the keyword lambda.

syntax:- `lambda arguments : expression`

lambda - keyword that starts the fun
argument - input parameter (like in normal fun).
expression - a single expression that gets evaluated and returned automatically.

* defined using a keyword lambda instead of def.
* It can take any number of arguments but must contain only one expression.
* expression is automatically returned (no need to use return).

Example:- Basic.

```
# Normal fun.
def add (a,b):
    return a+b.
# equivalent lambda func
add_lambda = lambda a,b : a+b
print (add_lambda (3,5))
```

o/p:- 8

* lambda with map():-

The map() function applies a function to every element in iterable.

```
num = [1,2,3,4,5]
squared = list (map(lambda x : x**2, num))
print (squared)
```

o/p:- [1, 4, 9, 16, 25]

* Lambda with filter() :-

The filter() fun filter elements from a list based on a condition.

```
num = [10, 15, 20, 25, 30]
# keep only num >= 20.
greater_than_20 = list (filter (lambda x : x >20 , num))
print (greater_than_20)
```

o/p :- [25, 30]

* Lambda with sorted ().

you can use lambda to define custom sorting logic.

```
data = [(1, 'apple'), (3, 'banana'), (2, 'cherry')]
# sort by second element (the first is name).
sorted_data = sorted (data, key = lambda x : x[1]).
print (sorted_data).
```

o/p :- [(1, 'apple'), (3, 'banana'), (2, 'cherry')]

* Lambda inside a function.

```
def make_multiplier (n):
    return lambda x : x*n.

double = make_multipliers (2)
triple = make_multipliers (3)

print (double (5))
print (triple (5))
```

o/p :- 10
15

* Conditional lambda.

Lambdas can include conditional expression too.

```
max_fun = lambda a, b : a if a>b else b.
print (max_fun (10, 20)).
```

o/p :- 20.

Concept	example	description.
Basic lambda	lambda x : x+1	simple inline func
map()	map (lambda x : x**2, nums)	Apply to each element.
filter()	filter (lambda x : x>10, nums)	filter elements.

sorted() sorted(items, key = lambda x: x[1]) custom sorting.

Returning lambda return lambda x: x*n create function factories

Recursive function:-

A recursive function is a function that calls itself to solve a smaller version of the same problem.

It must have a base condition to stop the recursion - otherwise, it will keep calling itself forever (causing an error).

Syntax:-

```
def function_name(parameter):
    if base_condition:
        return result
    else:
        # recursive call.
        return function_name(modified_parameter).
```

Ex:- factorial of a number.

```
def factorial(n):
    if n==1:        # base cond.
        return 1
    else:
        return n* factorial(n-1)   # recursive call

print(factorial(5))
```

o/p:- 120

Explain:-

factorial(5)
= 5 * fact(4)
= 5 * 4 * fact(3)
= 5 * 4 * 3 * fact(2)
= 5 * 4 * 3 * 2 * fact(1)
= 5 * 4 * 3 * 2 * 1 = 120.

* fibonacci series:-

```
def fibonacci(n):
    if n<=1:        # base cond.
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)   # recursive call
```

o/p:- 0 1 1 2 3 5

```
for i in range (6):
    print (fibonacci (i), end = " ").
```

* Countdown Time :-

```
def countdown (n):
    if n==0:    # base condi
        print ("Time's up!")
    else:
        print (n)
        countdown (n-1)  # recursive call.
countdown (5).
```

o/p :- 5 4 3 2 1 Time's up!

explain :- * The function calls itself with n-1 each time.
* when n becomes 0, its stop (base cond).

*
```
def fact (n):
    if n==1:
        return 1
    else:
        return n* fact (n-1)
fact (10).
```

o/p :- 3628800

* Fibbonaci Series :-

The fibbanaci series is a sequence of number where each number is the sum of the two preceding ones, starting from 0 and 1.

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

formula :- $F(n) = F(n-1) + F(n-2)$

$F(0) = 0$, $F(1) = 1$

ex:- using for loop :-

```
n = input int (input ("enter the num of terms :") #take i/p from user
a,b =0,1  # 1st, 2nd term
print ("fibonaci series:") # print message.
for i in range (n):  # use a for loop to generate the series
    print (a, end = " ") # print the current term
```

a,b = b, a+b. # calculate the next term, and update.

o/p:- Enter the num of terms - 7
fibonacci series : 0 1 1 2 3 5 8.

explain:- * n = int (input ("enter the num of term:"))
Ask the user how many num they want in the fibonacci series.
ex:- if you enter 7, you'll get 7 num in the series.

* a,b = 0,1
we start with two num.
a = 0 → 1st term
b = 1 → 2nd term

* for i in range (n):
This loop runs n times to print each fibonacci num

* print (a, end = " ")
print the current number (a) in the same line.

* a,b = b, a+b
This is the main logic of fibonacci:
- The next num is the sum of the previous two.
- After one iteration:
a takes value of b.
b takes the value of a+b.

ex:- when n=5.

iteration	a	b	o/p printed	next a,b values.
start	0	1	0	a=1, b=1
1	1	1	1	a=1, b=2
2	1	2	1	a=2, b=3
3	2	3	2	a=3, b=5
4	3	5	3	a=5, b=8.

* we start with a=0, b=1.
- Then we print a.
- calculate the next term c=a+b
- shift the var:- a become old b
b become c.
- Repet this for n times.

* using while loop:-

```
n = int(input("enter the limit:")) # take i/p from the user
a,b = 0,1  # initialize the first two num.
count = 0.  # initalize a counter
while count < n:  # start while loop
    print(a, end = " ")  # print the current fibonaci num
    a,b = b, a+b  # update 'a' and 'b' for the next num.
    count += 1  # increase the count by 1.
```

exp:- enter the limit : 7.

0 1 1 2 3 5 8.

exp:- * n = int(input("enter the limit:"))

- the user enters how many num they want to print.
- ex:- if the user enter 7, we'll print 7 fibonaci num.

* Initialize 1st two nums.

a,b = 0,1

- fibanacci series always start with 0 and 1.

a → first term
b → 2nd term.

* initalize counter.

count = 0

- we use count to keep track of how many num are printed
- the loop will countinue untill count reaches n.

* start the while loop.

while count < n;

- this loop runs as long as count is less than n.

so if n=7 the loop runs 7 times.

* print and update.

print(a, end = " ") → print the current num in one line.

a,b = b, a+b → - move b into a
- calculate the new b as the sum of old a+b.

count += 1.

↓

increase the counter by 1.

* Recursion:-

```
def fib(n):  # Define a recursive fun.
    if n <=1:  # Base case.
        return n
    else:
        return fib(n-1) + fib(n-2) # sum of previous two fibonacci nom
terms = int(input("enter the num of terms :")) # Take user i/p
print("fibonaci series:")
for i in range(terms): # print fibonacci num
    print(fib(i), end = " ")
```

explain:- Define fib(n)

```
def fib(n)
    if n<=1:
        return n
    else:
        return fib(n-1) + fib(n-2).
```

if n is 0 → retur 0
if n is 1 → return 1
other wise → return fib(n-1) + fib(n

* loop through num

```
for i in range(terms):
    print(fib(i), end = " ")
```

- call fib(0), fib(1), fib(2). ... up to fib(n-1)
- prints each fib num returned by the recursive fun.

* Break down for fib(5)

fib(5)
= fib(4) + fib(3)
= (fib(3) + fib(2)) + (fib(2) + fib(1))
= ((fib(2) + fib(1)) + fib(2)) + (fib(1) + fib(0))

- The continue untill it reaches fib(1) and fib(0) - the base case.

dp :- n = 7

0, 1, 1, 2, 3, 5, 8.

- Recursive fibbonacci is easy to understand, but slow for large n, because it recalculates the same value, many times.
- To fix that we can use memoization (storing previously calculated result).