Day 7 - Functions & Functional programming

Q What is a function?

→ A function is a block of code which is reusable that performs a specific task.

In simple words,

Instead of writing the same code again & again, we put it inside a function and we it whenever needed.

Q. Why Use functions?

→ - Avoid ~~repeatation~~ repetition
- Improve readability.
- Easy to debug
- Code reuse.

- Defining & Calling functions.

Defining a functions means ~~creating it and~~ creating it and calling a function means executing it.
A functions runs only when it is called, not when it is defined.

Function syntax:

```
def functionName():
    Code to be executed.
```

Here, function name can be anything, which will be later used to call the function.

Date
Page

- Calling a function & defining a function

- greeting function example:

→ Function definition:

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

greet()
```

greet() — Function name

print("Hello, welcome to Python") — code under function (inside function)

greet() — Calling the function

- Function Arguments

Arguments are values passed to a function so it can work with different data.

(1) Positional arguments

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

add(5, 3)
```

Here, we are passing two integers namely a and b variables and then the function prints the addition.

(2) Keyword Arguments

```
add(a=5, b=3)
```

Here, we are telling/declaring that, a is 5 and b is 3

(3) Default Arguments

```
def greet(name="User"):
    print("Hello", name)
greet()
greet("Piyush")
```

Here, if the argument is not passed (None in this case), then use the default value in the function (name="user" in this case).

- Return Values

The return statement sends a value back to where the function was called.
Example:

```
def square(num):
    return num*num
result = square(4)
print(result)
```

Date
Page

- print() vs return()

- print() → displays output
- return → provides output for reuse

→ Lambda Function

A lambda function is a small anonymous function written in one line.

Syntax:

```
lambda x : x * x
```

Example:

```
square = lambda x: x*x
print(square(5))
```

→ map() function

map() applies a function to each element of an iterable.

Example:

```
nums = [1, 2, 3, 4]
result = list(map(lambda x: x*2, nums))
print(result)
```

→ filter() function.

filter() selects elements that satisfy a given condition.
It helps remove unwanted data.
Example:

```
nums = [1,2,3,4,5]
even = list(filter(lambda x: x%2==0, nums))
print(even)
```

→ reduce() function.

reduce() combines all elements of an iterable into single value.

Example:

```
from functools import reduce
nums = [1,2,3,4]
total = reduce(lambda a, b: a+b, nums)
print(total)
```

Note: It is mainly used for aggregation tasks like sum, product or maximum.

→ Recursion

Recursion is a technique where a function calls itself. Useful for problems that can be broken into smaller identical problems. (subproblems)

• Important Parts

- Base condition (Stops recursion)
- ~~Recurv~~
- Recursive call (repeats logic)

Example:

```
def factorial(n):
    if n==1:
        return 1
    return n* factorial(n-1)
print(factorial(5))
```