In Python, a decorator is a function that takes another function as input and adds some extra functionality to it without modifying its original code. Decorators are a way to modify or enhance the behavior of a function or class without changing its source code.

In Python, decorators are written with the "@" symbol followed by the name of the decorator function. When a decorator is applied to a function or class, it wraps the original function or class with another function that provides the additional functionality.

For example, here's a simple decorator that adds a timer to a function:

import time

def timer\_decorator(func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

print(f"Elapsed time: {end\_time - start\_time} seconds")

return result

return wrapper

This decorator takes a function as input, and returns a new function that wraps the original function with a timer. To use it, simply apply the decorator to a function using the "@" symbol:

@timer\_decorator

def my\_function():

# do something

Now, whenever my\_function() is called, it will automatically print the elapsed time it took to run. This is just a simple example, decorators can be used for a wide range of purposes, such as logging, caching, authentication, and more.