**AI Day 07 Notes**

**Syed Mansoor ul Hassan Bukhari**

**Python Functions**

**Why Use Functions?**

Functions make the code less error-prone and more organized. A function is a block of code that performs a specific task whenever it is called. In larger programs, it is beneficial to create and use existing functions to keep the program flow organized and neat.

**Types of Functions**

1. **User-Defined Functions**: Created by the user as per their needs. It usually starts with def your\_function(argument).
2. **Built-in Functions**: Defined and pre-coded. Examples include min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.

**Function Arguments & Return Statement**

1. **Default Arguments**: We can provide a default value while creating a function. This way, the function assumes a default value even if a value is not provided in the function call for that argument.
2. **Keyword Arguments**: If we wish the order does not matter, then we use the keyword argument. We can provide arguments with key = value, this way the interpreter recognizes the arguments by parameter name. Hence, the order in which the arguments are passed does not matter.
3. **Required Arguments**: In this case, we must provide the value of arguments. If we don’t pass the arguments with a key = value syntax, passing should be in the exact positional order with the actual function definition.
4. **Arbitrary Arguments**: While creating a function, pass a \* before the parameter name while defining the function. The function accesses the arguments by processing them in the form of a tuple.

**Decorators**

Decorators in Python are a way to modify or extend the behavior of functions or methods without changing their code. They allow you to wrap another function or method in order to execute code before and/or after the wrapped function runs, or to modify its arguments or return value.

**Examples**

1. **User-Defined Function**: These are functions that users define themselves to perform a specific task.

def greet(name):

print("Hello, " + name + "!")

greet(“Mansoor Bukhari")

In this example, greet is a user-defined function that takes one argument, name, and prints a greeting message.

1. **Built-in Function**: Python provides several built-in functions like print(), len(), etc.

my\_string = "Hello, world!"

print(len(my\_string))

Here, len is a built-in function that returns the length of the string.

1. **Function with Default Arguments**: These functions assume a default value if a value is not provided in the function call for that argument.

def greet(name="World"):

print("Hello, " + name + "!")

greet()

In this example, if no argument is passed while calling the greet function, it uses “World” as the default value for name.

1. **Function with Keyword Arguments**: In these functions, arguments are identified by the parameter name, so the order of arguments doesn’t matter.

def full\_name(first, last):

print(first + " " + last)

full\_name(last="Salfi", first="Hacker")

Here, we’re calling full\_name function with keyword arguments. The order of arguments doesn’t matter because we’re specifying the values of first and last parameters by their names.

1. **Function with Arbitrary Arguments**: These functions can take any number of arguments.

def add(\*numbers):

return sum(numbers)

print(add(1, 2, 3, 4, 5))

In this example, add function takes any number of arguments and returns their sum. The \*numbers parameter in the function definition is used to collect all the extra arguments into a tuple.

1. **Decorators**: Decorators allow us to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it.

def my\_decorator(func):

def wrapper():

print("Something is happening before the function is called.")

func()

print("Something is happening after the function is called.")

return wrapper

@my\_decorator

def say\_hello():

print("Hello!")

say\_hello()

In this example, my\_decorator is a decorator that wraps the say\_hello function and modifies its behavior. The @my\_decorator line is a decorator syntax, which is equivalent to say\_hello = my\_decorator(say\_hello).