**PYTHON PROGRAMMING**

**ASSIGNMENT WEEK 3**

1.Mention the advantages of user defined functions.

Certainly! User-defined functions (UDFs) offer several advantages in programming and software development. Let’s explore them:

1. **Modularity and Code Organization**:
   * UDFs allow breaking down a large program into smaller, manageable blocks. Each function serves a specific purpose, making the code more organized and easier to understand.
   * By encapsulating functionality within functions, you achieve modularity, which simplifies maintenance and debugging.
2. **Code Reusability**:
   * Once you define a function, you can call it multiple times from different parts of your program.
   * This avoids duplicating the same code snippet throughout your application, reducing redundancy and promoting efficient code reuse.
3. **Readability and Understandability**:
   * UDFs improve program readability by dividing complex problems into smaller sub-programs.
   * Each function has a clear objective and interface, making it easier to comprehend the logic behind the program.
4. **Easy Debugging**:
   * When debugging, isolating faulty functions is straightforward.
   * You can focus on specific segments of code rather than dealing with an entire monolithic program.
5. **Independence from Source Code**:
   * UDFs can be modified independently of the main program source code.
   * This flexibility allows you to enhance or fix specific functionalities without affecting the entire application.
6. **Faster Execution**:
   * Once a function is defined, subsequent calls execute faster because the function’s logic is already compiled.
   * This efficiency contributes to overall performance.

 2. What is the purpose of def keyword?

The **def** keyword in Python serves the essential purpose of defining functions. When you encounter the **def** keyword, it signifies that you are creating a user-defined function. Here are some key points about the **def** keyword:

1. **Function Definition**:
   * The **def** keyword is placed before a function name, allowing you to create a logical unit of code.
   * A function consists of a sequence of statements indented under this name using the **def** keyword.
2. **Creating User-Defined Functions**:
   * You can define your own functions using the **def** keyword.
   * These functions can perform specific tasks or calculations based on your requirements.
3. **Code Reusability**:
   * By defining functions, you promote code reusability.
   * Instead of writing the same piece of code repeatedly, you encapsulate it within a function using the **def** keyword.
4. **Function Syntax**:
   * The syntax for defining a function is as follows:
   * def function\_name(parameters):
   * # Function definition statements
5. **Examples**:
   * Let’s explore some examples to illustrate the use of the **def** keyword:
     + Creating a simple function that prints “Hello”:
     + def python\_def\_keyword():
     + print("Hello")
     + python\_def\_keyword() # Output: Hello
     + Calculating the subtraction of two numbers:
     + def python\_def\_subNumbers(x, y):
     + return x - y
     + a = 90
     + b = 50
     + result = python\_def\_subNumbers(a, b)
     + print(f"Subtraction of {a} and {b} is = {result}") # Output: Subtraction of 90 and 50 is = 40
     + Generating the first 10 prime numbers:
     + def python\_def\_prime(n):
     + x = 2
     + count = 0
     + while count < n:
     + for d in range(2, x):
     + if x % d == 0:
     + x += 1
     + else:
     + print(x)
     + x += 1
     + count += 1
     + n = 10
     + print("First 10 prime numbers are:")
     + python\_def\_prime(n)
     + # Output: First 10 prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 27
     + Calculating the factorial of a number:
     + def python\_def\_factorial(n):
     + if n == 1:
     + return n
     + else:
     + return n \* python\_def\_factorial(n - 1)
     + num = 6
     + if num < 0:
     + print("Sorry, factorial does not exist for negative numbers")
     + elif num == 0:
     + print("The factorial of 0 is 1")
     + else:
     + print(f"The factorial of {num} is {python\_def\_factorial(num)}")
     + # Output: The factorial of 6 is 720
6. **Additional Notes**:
   * The **def** keyword is also used for defining methods within classes and special member functions like \_\_init\_\_().

3.Write about function prototype and its categories?

Function prototype tells the number of arguments passed to the function. Function prototype tells the data types of each of the passed arguments. Also, the function prototype tells the order in which the arguments are passed to the function.

It is categorised in five steps:

a) Function without arguments and without return type.  
b) Function with arguments and without return type.  
c) Function without arguments and with return type.  
d) Function with arguments and with return type.  
a) Function without arguments and without return type.  
o In this type no argument is passed through the function call and no

output is return to main function.  
o The sub function will read the input values perform the operation and print the result in the same block.  
b) Function with arguments and without return type.  
o Arguments are passed through the function call but output is not return to the main function.  
c) Function without arguments and with return type.  
o In this type no argument is passed through the function call but output is return to the main function.  
d) Function with arguments and with return type.  
o In this type arguments are passed through the function call and output is return to the main function.  
Example - Without Return Type  
Without argument With argument  
def add():  
a=int(input("enter a"))  
b=int(input("enter b"))  
c=a+b  
print(c)  
add()  
def add(a,b):  
c=a+b  
print(c)  
a=int(input("enter a"))  
b=int(input("enter b"))  
add(a,b)  
OUTPUT:  
enter a 5  
enter b 10  
15  
OUTPUT:  
enter a 5  
enter b 10  
15  
Example: With return type  
Without argument With argument  
def add(): def add(a,b):  
c=a+b  
return c  
a=int(input("enter a"))  
b=int(input("enter b"))

  What is lambda in Python? Why is it used?

Lambda functions are similar to user-defined functions but without a name. They're commonly referred to as anonymous functions.

Lambda functions are efficient whenever you want to create a function that will only contain simple expressions – that is, expressions that are usually a single line of a statement. They're also useful when you want to use the function once.

Differentiate between lambda and regular function.

In python, [def](https://www.geeksforgeeks.org/python-def-keyword/) defined functions are commonly used because of their simplicity. The def defined functions do not return anything if not explicitly returned whereas the lambda function does return an object. The def functions must be declared in the namespace. The def functions can perform any python task including multiple conditions, nested conditions or loops of any level, printing, importing libraries, raising Exceptions, etc.

Lambda functions are similar to user-defined functions but without a name. They're commonly referred to as anonymous functions.

Lambda functions are efficient whenever you want to create a function that will only contain simple expressions – that is, expressions that are usually a single line of a statement. They're also useful when you want to use the function once.

Define string and mention its operations.

The string operations include concatenation, scanning, substringing, translation, and verification. String operations can only be used on character, graphic, or UCS-2 fields. The CAT operation concatenates two strings to form one.