1.In Python, what is the difference between a built-in function and a user-defined function? Provide an example of each.

- A built-in function is a function that is already defined in the Python programming language. These functions are available for you to use without having to define them yourself. Some examples of built-in functions include print(), len(), and type(). A user-defined function is a function that you define yourself. You can define a user-defined function using the def keyword.

- For example, the following code defines a function called my_function():

- def my_function():
  print("Hello, world!")

- To call a user-defined function, you simply use its name followed by parentheses. For example, the following code calls the my_function() function

- my_function()

- The main difference between a built-in function and a user-defined function is that a built-in function is defined by the Python programming language, while a user-defined function is defined by the programmer.

2. How can you pass arguments to a function in Python? Explain the difference between positional

arguments and keyword arguments.

- There are two ways to pass arguments to a function in Python:
- positional arguments and keyword arguments. Positional arguments are arguments that are passed to a function in the order that they are defined in the function definition. For example, the following function takes two positional arguments, x and y:

- def my_function(x, y):
  print(x + y)

- To call this function, you would pass the arguments in the same order that they are defined in the function definition:

- my_function(1, 2)

- Keyword arguments are arguments that are passed to a function by name. For example, the following function takes two keyword arguments, x and y:

- def my_function(x, y):
  print(x + y)

- To call this function, you would pass the arguments in the following order:

- my_function(x=1, y=2)

- The main difference between positional arguments and keyword arguments is that positional arguments must be passed in the order that they are defined in the function definition, while keyword arguments can be passed in any order.

3. What is the purpose of the return statement in a function? Can a function have multiple return

statements? Explain with an example.

- The return statement in a function is used to exit the function and return a value. The value that is returned can be any type of data, such as a number, a string, or a list
- . A function can have multiple return statements, but only one of them will be executed. The first return statement that is encountered will be executed, and the function will then exit. Here is an example of a function with multiple return statements:

- def my_function(x, y):
  if x > y:
  return x
  else:
  return y

- In this function, there are two return statements. The first return statement is executed if x is greater than y. The second return statement is executed if x is less than or equal to y.
- The return statement is an important part of any function. It allows you to control the flow of your code and to return values from your functions.

4. What are lambda functions in Python? How are they different from regular functions? Provide an

example where a lambda function can be useful.

- Lambda functions are anonymous functions that can be used as arguments to other functions. They are defined using the lambda keyword, followed by the parameters in parentheses, and then the expression to be evaluated. For example, the following lambda function adds two numbers together:

- lambda x, y: x + y

- Lambda functions are useful when you need a function for a short period of time, or when you need to pass a function as an argument to another function. For example, the following code uses a lambda function to sort a list of numbers:

- ```
  numbers = [1, 2, 3, 4, 5]
  numbers.sort(key=lambda x: x * x)
  print(numbers)
  ```

- The lambda function in this code takes a number as an argument and returns the square of that number. This function is then used as the key function for the sort() method, which sorts the list of numbers in ascending order by their squares.
- Lambda functions are a powerful tool that can be used to simplify your code. They are especially useful when you need to write short, one-off functions

5. How does the concept of "scope" apply to functions in Python? Explain the difference between local

scope and global scope.

- The concept of scope in Python refers to the visibility of variables within a program. Variables that are declared inside a function are only visible within that function, and are said to be in local scope. Variables that are declared outside of any function are
  ```

visible to all functions in the program, and are said to be in global scope. The following code shows an example of local and global scope:

- x = 10 # global variable

- def my_function():
  y = 20
  print(x)
  print(y)

- my_function()

- In this code, the variable x is declared outside of any function, so it is a global variable. The variable y is declared inside the my_function() function, so it is a local variable.
- When the my_function() function is called, the value of the global variable x is printed, and then the value of the local variable y is printed. It is important to understand the difference between local and global scope when writing Python code. If you try to access a variable that is not in scope, you will get an error.

6. How can you use the "return" statement in a Python function to return multiple values?

- The return statement in Python can be used to return multiple values from a function. To do this, you simply use the comma operator to separate the values that you want to return. For example, the following function returns two values:

- def my_function():
    return 1, 2

- When this function is called, it will return the values 1 and 2. You can then use these values in your code. For example, you could assign them to variables:

- a, b = my_function()

- Now the variables a and b will contain the values 1 and 2, respectively. You can also return multiple values from a function using a tuple. A tuple is a data type that can hold multiple values. To return a tuple from a function, you simply use the tuple constructor:

- def my_function():
  return (1, 2)

- When this function is called, it will return the tuple (1, 2). You can then use this tuple in your code. For example, you could unpack it into variables:

- a, b = my_function()

- Now the variables a and b will contain the values 1 and 2, respectively. Returning multiple values from a function can be useful when you need to return more than one piece of information from a function. For example, you might want to return the result of a calculation and the error code.

7. What is the difference between the "pass by value" and "pass by reference" concepts when it

- **comes to function arguments in Python?**
- The arguments are passed by value. This means that when a function is called, a copy of the argument is passed to the function. Any changes made to the argument inside the function will not affect the original argument. For example, consider the following function:

- def my_function(x):
  x = x + 1
  print(x)

  my_function(5)

8. Create a function that can intake integer or decimal value and do following operations:

a. Logarithmic function (log x)

b. Exponential function (exp(x))

c. Power function with base 2 (2

x

)

d. Square root

- import math

```
x = float(input("enter the number"))


def log(x):
    return math.log(x)
print(log(x))


def exp(x):
     return math.exp(x)
print(exp(x))


def pow2(x):
   return math.pow(2, x)
print(pow2(x))


def sqrt(x):
     return math.sqrt(x*x)
print(sqrt(x*x))
```

9. Create a function that takes a full name as an argument and returns first name and last name.

- def split_name(name):
        first_name, last_name = name.split(" ")
        return first_name, last_name

```
print(split_name("bhavesh nikam"))
```