1. Why are functions advantageous to have in your programs?

Ans:- (1) Modularity: Functions allow you to break down complex tasks into smaller, manageable pieces of code. This makes your code more organized and easier to understand. Each function performs a specific task, and you can reuse these functions in different parts of your program or even in other projects.

(2)Reusability: As mentioned above, functions can be reused in different parts of your codebase or in other projects. This saves development time and effort, as you don't need to rewrite the same code over and over again.

(3) Readability: By using functions, you can give meaningful names to the different parts of your code, making it easier for yourself and other developers to understand the purpose of each block of code.

(4) Maintenance and Debugging: When a function is properly designed, changes or bug fixes can often be isolated to that specific function. This reduces the risk of introducing new bugs in other parts of the codebase, making maintenance and debugging more straightforward. (5) Code Organization: Using functions allows you to organize your code logically into smaller, focused units of functionality. This improves the overall structure of your program and makes it easier to navigate through the codebase.

2. When does the code in a function run: when its specified or when its called?

```
Ans:-
    When you define a function in a programming language, you are essentially creating a reusable block of
code that can be executed when called upon. The function definition specifies what the function does, but
it does not execute the code inside the function at that moment.
For Example:-
```

In [1]:

```
def greet():
    print("Hello, world!")

print("Before function call.")
greet()
print("After function call.")
```

```
Before function call.
Hello, world!
After function call.
```

3. What statement creates a function?

In [ ]:

```
Ans:-
    In most programming languages, including Python, the keyword "def" is used to create a function.
    For Example:-
```

In [2]:

```
def function_name(parameters):
    return some_value
def add_numbers(a, b):
    result = a + b
    return result
sum_result = add_numbers(2, 3)
print(sum_result)
```

```
5
```

4. What is the difference between a function and a function call?

```
Ans:-
   (1) Function:
A function is a reusable block of code that performs a specific task or set of tasks. It is designed to
encapsulate a piece of functionality that can be called and executed when needed. Functions are essential
for writing modular and organized code because they allow you to break down a large program into smaller,
manageable pieces.
For Example:-
```

In [3]:

```
def add_numbers(a, b):
    return a + b
```

```
(2) Function Call:
A function call, also known as a function invocation, is an action that instructs the program to execute
the code inside a specific function. When you call a function, the program jumps to the function's
definition, executes the code inside the function's body, and then returns back to the point in the code
from where the function was called.
For Example:-
```

In [4]:

```
result = add_numbers(5, 7)
print(result)
```

```
12
```

5. How many global scopes are there in a Python program? How many local scopes?

Ans:- (1) Global Scope: The global scope refers to the outermost level of a Python program. Variables defined in the global scope are accessible from any part of the program, including within functions. When you declare a variable outside of any function or block, it becomes a global variable. For Example:-

In [5]:

```
global_variable = 42

def function_example():
    print(global_variable)

function_example()
```

```
42
```

```
(2)Local Scopes:
Local scopes, also known as function scopes, are created when a function is called. Each function call
creates its own local scope, and any variables defined inside the function belong to that specific scope.
These variables are accessible only within the function and are not visible outside it.
For Example:-
```

In [6]:

```
def function_example():
    local_variable = 10  # This is a local variable
    print(local_variable)

function_example()
```

```
10
```

6. What happens to variables in a local scope when the function call returns?

Ans:- When a function call returns, the local variables defined within that function's scope are destroyed, and their values become inaccessible. This process is known as "cleaning up" the local scope. (1) Variable Destruction: As soon as the function call completes its execution (either by reaching the end of the function or encountering a return statement), the local scope associated with that function is destroyed. This means that all the local variables declared inside the function are removed from memory. (2) Inaccessibility: Once the local scope is destroyed, any attempt to access the local variables outside the function will result in an error. The variables no longer exist in the program's memory. For Example:-

In [7]:

```
def example_function():
    x = 10
    y = 20
    result = x + y
    return result

sum_result = example_function()
print(sum_result)
```

```
30
```

7. What is the concept of a return value? Is it possible to have a return value in an expression?

```
Ans:-
    In computer programming, a return value refers to the value that a function or a method gives back to
the caller after it has been executed. When you call a function, it performs a specific task and may
produce a result that can be used by the calling code. The return value is the data or information that is
passed back to the caller.Yes, it is possible to have a return value in an expression. In many programming
languages, functions can be used within expressions, and their return values can be directly utilized in
calculations or assignments.
```

In [8]:

```
def add_numbers(a, b):
    return a + b

result = add_numbers(5, 7)
print(result)
```

```
12
```

8. If a function does not have a return statement, what is the return value of a call to that function?

Ans:- If a function does not have a return statement, the return value of a call to that function is None in most programming languages. None is a special value that represents the absence of a meaningful value. When the function is called, and there is no explicit return statement that specifies a value to be returned, the function will implicitly return None.

In [9]:

```
def no_return_function():
    print("This function does not have a return statement.")

result = no_return_function()
print(result)
```

```
This function does not have a return statement.
None
```

9. How do you make a function variable refer to the global variable?

```
Ans:-
    In Python, if you want to make a function variable refer to a global variable, you can use the global
keyword inside the function. This way, the function will treat the variable as a global variable rather
than a local one.
    (1) Define the global variable outside of the function.
    (2)Inside the function, use the global keyword followed by the name of the variable you want to refer
to as a global variable.
    (3) Make sure to modify or access the global variable within the function.
    For Example:-
```

In [10]:

```
# Global variable
global_variable = 10

def modify_global_variable():

    global global_variable
    global_variable += 5  # Modifying the global variable inside the function

def access_global_variable():
    global global_variable
    print("Global variable value:", global_variable)
print("Before modifying:", global_variable)

modify_global_variable()
access_global_variable()
print("After modifying:", global_variable)
```

```
Before modifying: 10
Global variable value: 15
After modifying: 15
```

10. What is the data type of None?

Ans:- In Python, the data type of None is NoneType. None is a special object used to represent the absence of a value or a null value. It is often used to indicate that a variable or a function does not return any meaningful value. For Example:-

In [11]:

```
print(type(None))
```

```
<class 'NoneType'>
```

11. What does the sentence import areallyourpetsnamederic do?

Ans:- The sentence "import areallyourpetsnamederic" is not a complete sentence in natural language; it looks like a line of code in a programming context. It seems to be an attempt to use the "import" statement in Python, a popular programming language, followed by the module or package name "areallyourpetsnamederic."

12. If you had a bacon() feature in a spam module, what would you call it after importing spam?

In [ ]:

```
Ans:-If you had a bacon() feature in a spam module and you wanted to call it after importing the spam module
```

13. What can you do to save a programme from crashing if it encounters an error?

Ans:- When a program encounters an error, it can crash and abruptly stop executing. To prevent this from happening, we use error handling techniques in Python. One common approach is to use try-except blocks.

14. What is the purpose of the try clause? What is the purpose of the except clause?

```
Ans:-
    (1)Purpose of the try clause:
    The try clause is used to enclose a block of code where an exception might occur. It is a way to
define a risky section of code that may raise an exception. The code inside the try block is executed, and
if any exception occurs, the execution of the try block is immediately stopped, and the control is
transferred to the appropriate except block.
For Example:-
    try:
    # Code that might raise an exception
    # ...
except SomeException:
    # Code to handle the exception
    # ...
```

```
    (2)Purpose of the except clause:
    The except clause follows the try block and specifies what to do if a particular exception is raised
    during the execution of the try block. It allows you to define the handling logic for specific types
    of exceptions. If an exception occurs within the try block and the exception type matches the one sp
    ecified in the except clause, then the code inside the except block is executed.
```

For Example:-

In [14]:

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Invalid input. Please enter a valid number.")
```

```
Enter a number: 5
Result: 2.0
```

In [ ]:

```

```