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

**Ans:** Functions in Python provide several key benefits. They enable code organization by breaking tasks into manageable parts, making the code readable and easy to understand. Functions also enhance code reuse and modularity, saving time and effort. They aid in debugging by isolating specific functionality, and their encapsulation of variables prevents unintended interactions. Functions promote collaboration by allowing different parts of a program to be developed separately, and they contribute to better performance through optimized, reusable code segments.

**2. When does the code in a function run: when it’s specified or when it’s called?**

**Ans:** The code within a function in Python runs when the function is called. Defining a function does not execute the code inside it. The function's body contains the set of instructions that will be executed only when the function is invoked or called.

**3. What statement creates a function?**

**Ans:**The **def statement** creates a function in python. The **def** statement is used to define a function by specifying its name, parameters, and the block of code that constitutes the function's body.

Example:

def MyFirstFunction():

print("this is our 3rd week")

MyFirstFunction()

this is our 3rd week

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

**Ans: Function:** A function is a named block of code that performs a specific task or a set of tasks. It encapsulates a sequence of instructions and is designed to be reusable. Functions help in organizing code, promoting modularity, and making the code more readable and maintainable.

**Function Call:** A function call is the act of actually invoking or using a function that has been defined. It's the mechanism by which you instruct the program to execute the code inside the function. When you call a function, you provide the required arguments or parameters that the function expects.

In previous example

The segment

def MyFirstFunction():

print("this is our 3rd week")

and the segment

MyFirstFunction()

this is our 3rd week

is function call.

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

1. **Ans: Global Scope:** The global scope refers to the outermost level of the program. Variables defined in the global scope are accessible throughout the entire program, including within functions and sub-scopes. The global scope is established when you start your program and remains active until the program finishes.
2. **Local Scopes:** Local scopes are created whenever a function is called. Each function call creates its own local scope. Variables defined within a function are local to that function and can only be accessed within that function's block of code. When the function call completes, its local scope is destroyed, and the variables defined within it are no longer accessible.

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

**Ans:** When a function call returns in Python, the variables that were defined within the local scope of that function cease to exist. They are automatically deallocated or removed from memory. This process is known as variable scope and lifetime management.

Here's what happens step by step:

1. **Local Scope Creation:** When a function is called, a local scope is created specifically for that function. Any variables defined within the function's block of code are allocated memory within this local scope.
2. **Variable Assignment:** Variables within the local scope are assigned values and used for computations within the function.
3. **Function Call Completion:** When the function call completes (either because it reaches its end or due to a return statement), the local scope is terminated. This means that the memory allocated for the variables in that local scope is released.
4. **Variable Deallocation:** As the local scope terminates, the variables defined within it are deallocated. They are no longer accessible or valid outside of that specific function call.

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

**Ans:** The concept of a return value in programming refers to the value that a function provides back to the caller after it has completed its execution. When a function is called, it can perform some computation and then use the return statement to send a value (or multiple values) back to the caller. This returned value can be used for further computations, assignments, or any other operations.

Regarding the second question, a return value is not an expression in itself, but it can be used within expressions. The return value acts like any other value in expressions, allowing to perform operations with it.

Example

def mul(a,b):

result=a \* b

return result

multiplication\_result=mul(5,7)

print(multiplication\_result)

35

#Return value in an expression

total=mul(5,3)+mul(4,5)

print(total)

35

**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, it implicitly returns a special value called ‘None’.’ None ‘is a built-in Python object that represents the absence of a value. When a function call completes and there is no explicit ‘return’ statement, the function automatically returns ‘None’ by default.

Keep in mind that even though a function without a ‘return’ statement does not explicitly return a value, it still performs its intended operations, such as printing output, modifying variables, or performing other tasks within its scope.

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

**Ans:** To make a function variable refer to the global variable with the same name, you can use the ‘global’ keyword within the function. This allows you to modify the global variable from within the function, rather than creating a new local variable with the same name.

Keep in mind that while using global variables is sometimes necessary, it's generally recommended to avoid modifying them within functions whenever possible, as it can make code less readable and maintainable. It's usually better to pass variables as arguments to functions and return values from functions, as it promotes better encapsulation and reduces dependencies on global state.

Example:

def modify\_global():

global globalvar #declaring that I am gonna use the global variable

globalvar=20 #modifying

modify\_global()

print(globalvar)

20

**10. What is the data type of None?**

**Ans**: The data type of ‘None’ in Python is called ‘NoneType’. It is a special built-in type that represents the absence of a value or a null value. ‘None’ is often used to indicate the lack of a meaningful result or the absence of a specific value in situations where a value is expected.

**11. What does the sentence import areallyourpetsnamederic do?**

**Ans:** The sentence "import areallyourpetsnamederic" does not have any special meaning in the context of the Python programming language. In Python, the import statement is used to import modules or packages that provide additional functionality to your program. However, "areallyourpetsnamederic" is not a valid module or package name in Python's standard libraries or commonly used third-party libraries.

**Code**

import areallyourpetsnamederic

Traceback (most recent call last):

File "<pyshell#16>", line 1, in <module>

import areallyourpetsnamederic

ModuleNotFoundError: No module named 'areallyourpetsnamederic'

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

**Ans**: If I had a bacon() function in a module named spam, and I wanted to call it after importing the spam module, I would use the following syntax:

import spam spam.bacon()

Here, spam is the module name, and bacon() is the function within the spam module. By using spam.bacon(), I'm accessing and calling the bacon() function that's defined in the spam module.

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

**Ans:** To prevent a program from crashing when it encounters an error, we can use exception handling. Exception handling allows to catch and manage errors gracefully, so the program can continue running or handle the error in a controlled manner. Python provides a ‘try’ and ‘except’ block for this purpose.

**Example:**

10/0

Traceback (most recent call last):

File "<pyshell#23>", line 1, in <module>

10/0

ZeroDivisionError: division by zero

#Saving the Programme

try:

result= 10/0

except ZeroDivisionError:

print("Error occured fordeviding with 0")

Error occured fordeviding with 0

Here multiple except blocks can be used to catch different types of errors and handle them accordingly. Using exception handling helps improve the robustness of your program and provides a way to gracefully handle unexpected situations without crashing.

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

**Ans:** The ‘try’ and ‘except’ clauses are part of Python's exception handling mechanism. They serve distinct purposes:

1. **Purpose of the ‘try’ Clause:** The ‘try’ clause is used to enclose the block of code that might potentially raise an exception. It allows to designate a specific portion of a code where errors might occur. If an error occurs within the ‘try’ block, the program will not immediately crash; instead, it will transition to the associated ‘except’ block.
2. **Purpose of the ‘except’ Clause:** The ‘except’ clause is used to specify what should be done when a specific exception occurs within the ‘try’ block. It defines the handling logic for that particular type of exception.

If an exception of the specified type (in this case, SomeException) is raised in the ‘try’ block, the code in the corresponding ‘except’ block will be executed. This allows you to gracefully handle errors, provide meaningful feedback, and continue program execution without crashing.

(The example used in the previous question can be used in this question as well)