ASSIGNMENT 3

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

Functions are advantageous to have in your programs for several reasons:

1. Modularity: Functions allow to break code into smaller, reusable pieces. This modular approach makes code easier to understand, maintain, and debug. We can focus on individual functions without needing to understand the entire program at once.

2. Reusability: Once define a function, it can call it multiple times from different parts of our program. This saves from duplicating code, which reduces the chances of errors and makes code more efficient.

3. Abstraction: Functions abstract away the implementation details. we can create a function with a meaningful name and a clear purpose. Users of the function don't need to know how it works internally, just its input and output requirements.

4. Readability: Well-named functions make our code more readable and self-documenting. Instead of writing a long sequence of operations in our main program, we can call a function with a descriptive name, making it easier for others to understand what the code does.

5. Testing: Functions are easier to test in isolation. We can create test cases specifically for each function, ensuring that it behaves correctly. This simplifies the debugging process because we can pinpoint issues to specific functions.

6. Scalability: As your program grows, functions allow you to scale your codebase without making it overly complex. You can add new functions for new features or improvements without affecting existing code.

7. Collaboration: In team programming, functions provide a way to divide the work among team members. Different team members can work on different functions independently, and then these functions can be integrated into the main program.

8. Error Handling: Functions can encapsulate error-handling logic. We can have functions that handle specific error cases, making your main program more focused on the core logic.

9. Code Organization: Functions help organize your code logically. They group related operations together, making it easier to locate and update specific functionality.

functions enhance code organization, readability, reusability, and maintainability, making your programs more efficient, reliable, and easier to work with. They are a fundamental building block in software development and promote good coding practices.

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

During defining a function, we are creating a reusable block of code with a specific name and a set of parameters. The code inside the function is not executed at the function definition. It only gets executed when you call the function using its name followed by parentheses, passing any required arguments.

Example

```
def say_hai():        # function definition
    print("hai")
say_hai()     #function call
```

3. What statement creates a function?

In function def statement is used to create function followed by function name and arguments with in parenthesis

```
def say_hai():        # function definition
    print("hai")
```

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

A function is a reusable block of code that performs a specific task or set of tasks. It is defined using the def keyword in Python When define a function specify its name, parameters (if any), and the code that should be executed when the function is called. Functions are like templates or blueprints for actions want to perform in program. Functions are not executed when defined; they are executed when called.

Example:

```
def add (a, b):
    result = a + b
    return result
```

Function Call: A function call is the act of executing a function by using its name followed by parentheses. When call a function, provide any required arguments or parameters that the function expects. The function call triggers the execution of the code defined within the function's body. After the function completes its execution, it can optionally return a value to the caller

Example of a function call:

```
result = add(3, 5)
```

In the example above, add is the function, and add(3, 5)is the function call. The function call provides the values 3and 5 as arguments to the add function, which then executes its code, adds the numbers, and returns the result.

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

In a Python program, there is typically one global scope and multiple local scopes.

Global Scope: There is only one global scope in a Python program. The global scope encompasses the entire program, and any variable defined outside of all functions or classes is considered part of this global scope. Variables defined in the global scope are accessible from any part of the program, including within functions.

Example:

```
var = 42  # This variable is in the global scope

def my_function():

    print(ver)  # Accessing the global_variable within the function

my_function()
```

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 part of its local scope and are not accessible outside of that function, except through return values.

Example:

```
def my_function():

    local_var = 10        # This variable is in the local scope of my_function

    print(local_var)

my_function()

print(local_var)        # This would result in an error because local_var is not defined in the global scope
```

In addition to global and local scopes, Python also has non-local scopes for nested functions, but those are not counted as separate global or local scopes. Non-local scopes allow inner functions to access variables from their containing (enclosing) functions.

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

allocated memory when the function is called are destroyed when the function exits or returns.so that the local variable are not accessible after function return or exist

Variable Lifetime: Local variables are temporary and have a limited lifetime that corresponds to the execution of the function. They are created when the function is called and destroyed (or "garbage collected") when the function returns. This means that the local variables cannot be accessed or referenced outside the function after the function has finished executing.

Memory De-allocation: When the function returns, Python releases the memory occupied by the local variables, making that memory available for other uses in the program.

Here's an example to illustrate this:

```
def my_function():
    local_variable = 42              # This variable is in the local scope of my_function
    print("Inside the function:", local_variable)
my_function()                        # Calling the function
                                     # The local_variable is no longer accessible here; it has gone out of scope
                                     # Attempting to print it here would result in an error
```

In the example above, local_variable exists only within the my_function function's local scope. Once the function call my_function() returns, the local scope is destroyed, and local_variable is no longer accessible or valid.

If you need to retain or use the value of a variable from a function's local scope after the function has returned, you can do so by returning the value from the function and storing it in a variable in a higher (e.g., global) scope or by passing it as an argument to another function.

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

The concept of a return value in programming refers to the value that a function provides as output when it completes its execution. Functions can return values to the caller. These return values can then be used in expressions, assigned to variables, or passed as arguments to other functions.

Here's how the concept of a return value works:

Function Return Value: When a function is defined then specify that it returns a value using the return statement. The return statement is used to pass a value back to the caller of the function.The returned value can be of any data type (e.g., integer, string, list, custom object).

```
  def add(a, b):
      result = a + b
      return result        # This function returns the value of result
```

Using Return Values: When you call a function that returns a value, you can capture that value by assigning it to a variable or using it directly in an expression.Return values are often used to perform further computations or make decisions based on the result of the function.

Return Value in Expressions : it is possible to use a return value in an expression. You can directly include the function call (which returns a value) within an expression to perform calculations or operations.

```
 def add(a, b):
      result = a + b
      return result        # This function returns the value of result
```

```
new_result = add(3, 5) * 2  # Using the return value of 'add' in an expression
```

In the example above, the return value of the add function is used within the expression add(3, 5) * 2. This allows you to perform calculations based on the value returned by the function.

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

If a function does not have a return statement, or if it exits without encountering any return statement, the function will return a special value called None. None represents the absence of a value or the lack of a return value.

example:

```
def my_fun():
    print("This function does not have a return statement")
result = my_fun():
print(result)                    # This will print None to the console
```

In this example, the my_fun() function does not contain a return statement. When you call this function, it prints a message but does not explicitly return any value. Consequently, the value assigned to the variable result is None, and if you print it, you'll see None in the output.

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

a function variable refers to a global variable by using the global keyword within the function. This keyword indicates that a variable with the same name as a global variable should be treated as a global variable, even if it's assigned a value within the function. This allows to modify the global variable's value from within the function.

Example

```
global_variable = 10                    # This is a global variable
def modify_global_variable():
    global global_variable              # Declare that 'global_variable' should be treated as global
    global_variable = 20                # Assign a new value to the global variable
modify_global_variable()                # Call the function to modify the global variable
print(global_variable)                  # Prints 20 because the global variable was modified by the function
```

In this example, global_variable is a global variable with an initial value of 10. Inside the modify_global_variable function, we declare global global_variable, indicating that we want to modify the global variable, not create a new local variable with the same name. The function assigns a new value (20) to the global variable, and when we print it outside the function, we see the modified value.

10. What is the data type of None?

In Python, None is a special constant that represents the absence of a value or a null value. It is not a specific data type, but rather it belongs to the "NoneType" type, which is a data type of its own

```
value = None

print(type(value))                # This will print <class 'NoneType'>
```

11. What does the sentence import areallyourpetsnamederic do?

Import areallyourpetsnamederic do importing the class areallyourpetsnamederic. import statements are used to bring in external modules or libraries to extend the functionality of program.

If attempt to run or execute the sentence "import areallyourpetsnamederic," Python will raise an ImportError if it doesn't recognize "areallyourpetsnamederic" as a valid module or library. Areallyourpetsnamederic does not have any built-in meaning in Python

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

```
Import spam

spam.bacon()
```

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

To prevent a program from crashing when it encounters an error, using error handling techniques. Python provides several mechanisms for handling exceptions (errors), allowing you to gracefully handle unexpected situations and avoid program crashes. Use try and except blocks to catch and handle exceptions.

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

the try and except clauses are used together to implement error handling. The try clause is used to enclose a block of code that might raise an exception (error).It defines the "try" section where the code is executed, and the interpreter checks for exceptions during its execution. If an exception occurs within the try block, the normal flow of execution is interrupted, and the program looks for an associated except block to handle the exception. Multiple except blocks can be used to catch different types of exceptions and handle them differently

```
try:                              # Code that might raise an exception
    result = 10 / 0
except ZeroDivisionError          # Handle the exception
    print("Division by zero is not allowed.")
```