1. Functions are advantageous can be explained through following program in python:

```
def rectangle_area(len,bred) :          #   function declaration/definition
return len*bred

def rectangle_perimeter(len,bred) :
return 2*(len+bred)

len=int(input("Enter the length :  "))    # taking input from user
bred=int(input("enter the breadth:  "))

area= rectangle_area(len,bred)          # function call for calculating area of rectangle
print("Area :  ", area)

perimeter= rectangle_perimeter(len,bred)   # function call for calculating perimeter
print("Perimeter : ",perimeter)
```

i) **Reusability :** As shown in above code, functions **'rectangle_area()'** and **'rectangle_perimeter'** can be used multiple times with different input values

ii) **Modularity:** function makes code more organized and manageable. Program can be broken down into smaller set of tasks, modules and functions which are itself manageable . in the above code, functions to calculate perimeter and area of rectangle are defined separately

iii) **Readability:** The function names (**'rectangle_area()**' and **'rectangle_perimeter()'**) clearly convey their purpose, making the code more readable and self-explanatory

iv) **Abstraction**: The implementation details of the calculations are hidden within the functions. Users only need to provide the required inputs and use the returned values, without worrying about the underlying logic**.**

v) **Testing and Debugging:** The functions can be individually tested and debugged to ensure their correctness. If an issue occurs, it is easier to identify and fix problems within the specific function rather than searching through the entire code

2. The code in a function runs when the function is called.
   For example, consider the following code :

```
def func():
   print("Hello")

print("Before function call")

func()

print("After function call")
```

   In the above code, the function **func()** is defined, which simply prints "**Hello".** The program flow starts from the top and executes the code sequentially.
   When code goes to the line func(), it is a function call, and at that point, the code inside the 'func()' function is executed. It will print "Hello". After executing the function, the program flow returns to the point immediately after the function call and continues executing the remaining code. In this case, it will print **"After function call".** so the output of the above code will be :

```
Before function call
Hello
After function call
```

3. The **def** statement is used to create a function
   The def statement, short for "**define**," is followed by the name of the function and a pair of **parentheses ().** The parentheses may include any parameters or arguments that the function accepts. After the parentheses, a **colon :** is used to indicate the start of the function's code block
4. **Function** is the defined block of code, and the **function call** is the usage of that function to execute its code with specific arguments or parameters.

```
def name(user):
  print("hello" + name)
name(Vipin)    # function call
```

   here **name()** is the function which takes **'user'** parameter and line 'name(vipin)' is function call

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

   Global scope:- The global scope exists throughout the entire program, and any variables or functions defined in the global scope can be accessed from any part of the program

   Local scope :- Local scopes, also known as function scopes, are created whenever a function is called. Each function call creates its own local scope. Variables declared in local scope are accessible within their local scope

6. When a function call returns, the local variables within that function's local scope are destroyed or deallocated, and their values are no longer accessible. Any variables defined within the local scope are no longer accessible outside of the function.

7. Return value refers to the value that a function can send back or "return" to the code that called it. When a function is called, it may perform certain operations and produce a result, the return value allows the function to communicate that result back to the caller.
   A function can use the **'return'** statement to specify the value it wants to return. The return statement is followed by an expression or value that will be evaluated and sent back as the return value

```
def addition(a, b):
    return a +b
result = addition(3, 5)
print("Sum:", result)
```

In the above code, the addition() function takes two arguments a and b and returns their sum using the return statement. The function call addition(3, 5) evaluates to 8, which is then assigned to the result

It is possible to have a return value directly in an expression in Python. Consider the following code :-

```
def multiplication(x, y):
   return x * y

result = multiplication(4, 6) + 2
print("Result:", result)
```

In this code, the return value of the **'multipliplication()'** function, which is the product of 4 and 6, is directly used in the expression **multiplication(4, 6) + 2**. The resulting value, 26, is then assigned to the result.

8. If a function does not have a return statement, the return value of a call to that function is **None**. 'None' is a special Python object that represents the absence of a value.

9. If you want to make a function variable refer to a global variable, you can use the **'global'** keyword within the function. The global keyword allows you to indicate that a variable inside a function should be treated as a global variable rather than a local variable

```
var = 10          # Global variable

def modify_global():
    global var              # Declare the variable as global
    var = 20                # Modify the global variable within the function
    print("Inside the function:", var)

print("Before function call:", var)
modify_global()
print("After function call:", var)
```

Output will be:
Before function call: 10
Inside the function: 20
After function call: 20

10. The data type of None is **NoneType**. NoneType is a special data type that represents the absence of a value.

11. The sentence "import **areallyourpetsnamederic"** does not have any standard meaning in Python. It is not a valid Python module or package, and attempting to import it would result in a '**ModuleNotFoundError**' unless you have a custom module or package named "areallyourpetsnamederic" in your Python environment

12. After importing the spam module, you can call the bacon() feature by using the syntax **spam.bacon().**

After calling it, 'The bacon()' function can then perform whatever task it is designed to do

13. To prevent a program from crashing when it encounters an error, you can use error handling techniques to gracefully handle exceptions
we use a combination of **'try'**, **'except'**, and **'finally'** blocks to handle exceptions.
The try block contains the code that might raise an exception
The except block specifies how to handle the exception if it occurs.
The finally block can be used to specify code that should be executed regardless of whether an exception occurred or not.
Consider the following code :-

```
try:
    # Code that might raise an exception
    num1 = 10
    num2 = 0
    result = num1 / num2
    print("Result:", result)
```

```
except ZeroDivisionError:
    # Handling specific exception(s)
    print("Error: Cannot divide by zero")

except Exception as e:
    # Handling any other exceptions
    print("An error occurred:", str(e))

finally:
    # Code that will always be executed
    print("Program execution completed.")
```

In the above code, the division **num1 / num2** inside the try block could potentially raise a **ZeroDivisionError** if num2 is zero. In such a case, the program would crash.
However, by using the try-except block, we catch the ZeroDivisionError and print a custom error message. The finally block is executed regardless of whether an exception occurred or not.

14. The purpose of the **try clause** is to enclose a block of code that may potentially raise an exception. It allows you to specify the code that could **potentially cause an error or exception.**
The **except clause** is used to define the specific actions or code that should be executed if an exception occurs within the corresponding try block.

In combination, the try and except clauses form a try-except block that enables you to implement error handling and exception management in your code.