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

Reduce code duplication. When you have a block of code that you need to run multiple times in your program, you can define a function for that block of code and then call the function each time you need to run it. This prevents you from having to copy and paste the same code multiple times, which can make your code more readable and easier to maintain.

Improve code readability and maintainability. Functions can help to improve the readability and maintainability of your code by grouping related code together and giving it a meaningful name. This makes it easier to understand what the code is doing and to make changes to it if necessary.

Make your code more reusable. Once you have defined a function, you can use it in other programs or scripts. This can save you time and effort, as you don't have to write the same code from scratch each time.

Improve debugging. When your program has a bug, it can be difficult to track down the cause of the bug if your code is long and complex. Functions can help to make your code easier to debug by breaking it down into smaller, more manageable pieces.

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

The code in a function runs when it is called, not when it is specified. When you specify a function, you are simply creating a blueprint for the function. The function is not actually created until it is called.

3. What statement creates a function?

The def statement in Python creates a function. The syntax for a function definition is as follows:

def function\_name(parameters):

body

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

A function is a block of code that performs a specific task. It is defined using the def keyword in Python. For example, the following code defines a function called add\_numbers():

def add\_numbers(a, b):

"""This function adds two numbers together."""

return a + b

A function call is an instruction to the computer to execute the code inside a function. It is written using the function's name, followed by parentheses containing any arguments that the function needs. For example, the following code calls the add\_numbers() function, passing it the numbers 1 and 2 as arguments:

result = add\_numbers(1, 2)

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

There is only one global scope in a Python program. The global scope is the top-level scope of the program, and it contains all of the variables that are defined at the top level of the program.

There can be multiple local scopes in a Python program. A local scope is created when a function is called, and it contains all of the variables that are defined inside the function. When a function returns, the local scope is destroyed.

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

When a function call returns, the local variables in the function's scope are destroyed. This means that the variables are no longer available to the code that called the function.

def add\_numbers(a, b):

"""This function adds two numbers together."""

return a + b

result = add\_numbers(1, 2)

print(result)

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

A return value is the value that is returned by a function when it is finished executing. A return value can be any Python object, including numbers, strings, lists, dictionaries, and functions.

Yes, it is possible to have a return value in an expression. An expression is a piece of code that evaluates to a value. When an expression is used in a return statement, the value of the expression is returned by the function.

For example, the following function returns the sum of two numbers:

def add\_numbers(a, b):

"""This function adds two numbers together."""

return a + b

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, the return value of a call to that function is None. This is because every function in Python returns something, even if it's nothing. The None object is a special object in Python that represents the absence of a value.

Here is an example of a function that does not have a return statement:

def no\_return():

pass

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

To make a function variable refer to the global variable, you can use the global keyword. The global keyword tells the Python interpreter that the variable is defined in the global scope, and not in the local scope of the function.

Here is an example of how to use the global keyword:

x = 10

def foo():

global x

x = 20

foo()

print(x)

10. What is the data type of None?

None is a special object in Python that represents the absence of a value. It is not the same as 0, False, or an empty string. None is a data type of its own (NoneType) and only None can be None.

# Assign None to a variable

x = None

# Check if a variable is None

if x is None:

print("x is None")

# Use None as a default value

def foo():

return None

# Call foo() and print the result

result = foo()

print(result)

11. What does the sentence import areallyourpetsnamederic do?

The sentence import areallyourpetsnamederic imports a module named areallyourpetsnamederic. This module is not a real Python module, but it is a good example of how to import a module.

When you import a module, the Python interpreter will look for a file with the same name as the module in the current directory and all of the directories that are listed in the PYTHONPATH environment variable. If the Python interpreter finds the file, it will load the module and make the functions and variables that are defined in the module available to your program.

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

To call the bacon() feature in a spam module after importing spam, you would use the following code:

import spam

bacon()

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

There are a few things you can do to save a program from crashing if it encounters an error.

Use try-except blocks. A try-except block allows you to handle errors gracefully. The try block contains the code that might cause an error, and the except block contains the code that will be executed if an error occurs. For example:

try:

number = 10 / 0

except ZeroDivisionError:

print("Error: Division by zero")

Use logging. Logging can be used to record errors that occur in a program. This can be helpful for debugging and troubleshooting problems. For example:

import logging

logger = logging.getLogger(\_\_name\_\_)

try:

number = 10 / 0

except ZeroDivisionError:

logger.error("Error: Division by zero")

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

The try clause is used to test a block of code for errors. If an error occurs, the except clause is executed. The else clause is executed if there is no error. The finally clause is executed regardless of whether or not there is an error.

try: The try block contains the code that you want to protect from errors. If an error occurs in the try block, the except block will be executed.

except: The except block contains the code that you want to run when an error occurs. The except block can be used to handle specific types of errors, or it can be used to handle all errors.

else: The else block contains the code that you want to run if there are no errors in the try block.

finally: The finally block contains the code that you want to run regardless of whether or not there is an error. The finally block is often used to close files or to release resources.