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

1. Avoid duplicate code:  
   Many times we need the same code running multiple times. Instead of having multiple copies of the same code, we can put that code in a function so that we can call it multiple times.
2. Less code maintenance:  
   When we avoid duplicate code by putting them into functions, we do not have to make changes to them at multiple places as the same code is not repeating or duplicate.
3. Easy to read and collaborate:  
   With a set of code grouped into a function, we can group code in a meaningful manner and add documentation explaining it. This also makes it easier for other users to understand the code and enable collaboration.
4. The code in a function will run only when it is called and not when it is defined.
5. We can have multiple scopes for variables as we can have a different scope to a variable within a function.
6. A function can be used as a part of an expression

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

The code inside a function is not executed when the function is defined. The code in a function will never run unless it is called or referenced.

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

Functions in Python are defined by the “def ” statement followed by the function name and parentheses () . Within the parentheses, we can pass variables into the function.

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

**A function is** just a piece of code defined using the “def” statement. This makes a function just a declaration. This code acts a blueprint for performing a task that is mentioned in it.

**A function call is** an action being performed of executing the code as defined within the function. Unless a function is called, it will not be executed. Thus a function call executes a function and performs all tasks mentioned in the function.

Thus a function is a blueprint to perform a task and a function call is the actual execution of the task based on the blueprint.

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

There are 4 types of scopes in python:

1. Local Scope  
   Local Scope is created at function call. So you can have **as many local scopes** as you have function calls.
2. Enclosing Scope
3. Global Scope  
   There **is only one global Python scope** per program execution. This scope remains in existence until the program terminates and all its names are forgotten.
4. Built in Scope : Pre-defined names in python such as print(), id()

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

When the function returns, the local scope is destroyed and the variable names are forgotten.

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

All python functions return a value when it completes executing. If the return value is not explicitly defined in a function then it is implicitly defined.

The Python return statement allows to send any Python object from within the function’s local scope back to the caller code.

It is possible to have an expression in the return statement that evaluates into a value. If a return statement is an expression, then the expression is evaluated and the value is returned. For example *return a + b*will return the addition of the 2 variables a & b.

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

In Python, every function returns something. If there is no return statement, then it returns **None**.

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

To make a function variable refer to a global variable we use the “global” keyword while declaring or referencing the variable in the function.

For example:

*a = 1*

*def b() :*

***global a***

*a += 1*

*return a*

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

None represents a null value, or no value at all. None is not the same as 0, False, or empty string. None is a data type of *NoneType*.

*>>> type(None)*

*<class 'NoneType'>*

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

The import statement imports the module named areallyourpetsnamederic

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

In the example below the bacon() function would continue to have the same name and I can reference it as :

*import spam*

*spam.bacon()*

Alternatively I can import only the function as :

*from spam import bacon()*

*bacon()*

If I use an alias then I will refer to the function with the alias name. If not, then I will continue to call it as bacon()

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

To avoid a program from crashing, I can use error handlers (try error handler) to capture the possible errors which could do the following:

1. Appropriately respond to the user.
2. Follow alternate course of action (except clause in the try error handler)
3. Perform a clean-up such as closing database connections, etc.
4. Perform logging in order to have logs for future debugging.

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

The try clause allows us to execute code that could possibly be erroneous during runtime safely without crashing the program.

Therefore if in case an error occurs during the runtime for the code that is within the try block the program will not directly crash but instead follow an alternative course of action as mentioned in the except clause.

Therefore the except clause will run only if there was an error that occurred within the try block. We can have multiple except clauses within a try error handler. Each except statement can test for a specific type of error.

In the except clause we can provide an alternative action that can be performed in case of an error occurred. Thus the program will not crash but proceed as normal because the error was handled.