Solutions

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

   Reusability: Functions allow you to write code once and reuse it multiple times in different parts of your program or in different programs. This can save you a lot of time and effort in coding.

   Modularity: Functions allow you to break down your program into smaller, more manageable pieces, which makes your code more organized, easier to read and understand, and easier to maintain.

   Abstraction: Functions can be used to hide complex or low-level details of your program behind a simple interface. This allows you to focus on the high-level logic of your program without getting bogged down in the details.

   Testing: Functions make it easier to test your code because you can isolate specific parts of your program and test them independently. This can help you catch and fix bugs more quickly and easily.

   Collaboration: Functions can make it easier to collaborate with others on a project because you can divide the work into smaller parts and assign each part to a different person.

2. The code in a function runs when the function is called, not when it is specified. Defining a function simply creates a named block of code that can be executed later. When the function is called, the code inside the function block is executed with the arguments passed to the function. Until the function is called, the code inside the function block is not executed. This means that you can define functions at the top of a program or module and call them later in the code. When the function is called, the code inside the function block is executed, and any actions or changes specified by the code are carried out.

3. The "def" statement is used to create a function in Python.

4. A function is a named block of code that performs a specific task when called. It takes in one or more input arguments, performs some operations on them, and can optionally return a result.

   On the other hand, a function call is the act of invoking a function by specifying its name and passing in any required arguments. When a function is called, the code inside the function's block is executed with the provided arguments, and any specified action or change is carried out.

5. In Python, there is only one global scope per program, and it is created when the program starts executing. The global scope is used to store variables and functions that are defined at the top level of the program and can be accessed from anywhere in the program.

   On the other hand, local scopes are created whenever a function is called. Each function call creates a new local scope, which is used to store variables that are defined within the function. Local scopes are destroyed when the function returns, and any variables defined in the local scope are no longer accessible

6. When a function call returns in Python, the local scope created for that function is destroyed, and any variables defined within that local scope are also destroyed. This means that the variables are no longer accessible from outside of the function and will be lost unless they have been returned from the function or their value has been stored somewhere else before the function returned.

7. In Python, a return value is the value that a function returns to the caller when it completes execution. The return statement is used to specify the return value of a function.

   For example, consider the following function:

```
def add_numbers(x, y):
    result = x + y
    return result
```

   This function takes two arguments, adds them together and returns the result using the return statement. When the function is called, the returned value can be assigned to a variable or used in an expression:

```
sum = add_numbers(5, 7)  # assigns the returned value to the 'sum' variable
print(sum)  # prints '12'

total = add_numbers(3, 4) * 2  # uses the returned value in an expression
print(total)  # prints '14'
```

8. If a function does not have a return statement, the function will not explicitly return a value. In this case, the return value of a call to that function will be None

9. In Python, you can make a function variable refer to the global variable by using the global keyword. When you use the global keyword inside a function, it tells Python that the variable you are referencing is a global variable, rather than a local variable.

10. None is a built-in constant object in Python, and it is considered to be a data type of its own, called the NoneType. The NoneType has only one value, which is None.

11. The sentence "import areallyourpetsnamederic" doesn't convey any specific meaning on its own. It appears to be a Python code statement that attempts to import a Python module named "areallyourpetsnamederic".

12. If you imported a module called "spam" which contains a feature called "bacon()", you can call the feature using the following syntax:

```
import spam
spam.bacon()
```

13. To save a program from crashing if it encounters an error, you can implement error handling in your code. Error handling allows you to gracefully handle errors and prevent your program from crashing by providing an alternative course of action when an error occurs. In Python, you can use a try-except block to implement error handling.

14. In Python, the "try" and "except" clauses are used for implementing error handling. The purpose of the "try" clause is to enclose a block of code that may raise an exception (an error).

    When this block of code is executed, Python checks for any exceptions that might occur. If an exception is raised within the "try" block, Python immediately jumps to the "except" clause to handle the exception.

    The purpose of the "except" clause is to provide an alternative course of action when an exception occurs within the "try" block. This alternative course of action might include logging the error, displaying a user-friendly message, or taking any other appropriate action to handle the error and prevent the program from crashing.