**Assignment 3**

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

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

Reusability: Functions allow you to write a block of code once and reuse it multiple times throughout your program. This saves time and effort, as you don't have to write the same code over and over again.

Modularization: Functions help break down a large program into smaller, more manageable parts. Each function performs a specific task, making it easier to understand and debug your code.

Abstraction: Functions can hide complex implementation details and provide a simple interface for other parts of the program to use. This allows other parts of the program to use the function without worrying about how it's implemented.

Maintainability: Functions make your code easier to maintain and update. If you need to change the behavior of a function, you can do so without affecting the rest of the program.

Testing: Functions are easier to test than a large block of code. By breaking your program down into smaller functions, you can test each function individually, ensuring that it works correctly before integrating it into the larger program.

Overall, functions help make your code more efficient, modular, and maintainable.

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 the function is called, not when it is specified.

When you define a function in Python, you are essentially creating a block of code that can be executed later on. This code is not executed until the function is called.

To call a function, you simply use the function name followed by parentheses. For example, if you have a function named my\_function, you would call it like this:

my\_function()

When this line of code is executed, the code inside the my\_function function will be executed. Any parameters passed to the function will be used as input, and the function will return its output.

So, to summarize, the code in a function is executed when the function is called, not when it is defined.

3. What statement creates a function?

To create a function in Python, you use the def statement.

The def statement is followed by the name of the function, a pair of parentheses that may include parameters, and a colon. The body of the function, which contains the code that will be executed when the function is called, is indented below the def statement.

Here is an example of a simple function named my\_function that takes no parameters and simply prints "Hello, world!":

def my\_function():

print("Hello, world!")

You can then call this function by using its name followed by parentheses:

my\_function() # Output: Hello, world!

Note that the def statement creates the function but does not execute its code. The code is executed only when the function is called.

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

A function is a block of code that performs a specific task when it is executed. It may or may not take input arguments, and it may or may not return a value as output. A function is defined using the def statement in Python.

On the other hand, a function call is an instruction to execute a function at a particular point in a program. A function call provides input arguments (if any) to the function, which processes the input and may return an output value. A function call is written by using the function name followed by parentheses containing the input arguments (if any).

To illustrate the difference between a function and a function call, consider the following example:

def add\_numbers(x, y):

return x + y

result = add\_numbers(2, 3)

print(result)

In this example, we define a function called add\_numbers that takes two input parameters and returns their sum. We then call this function with arguments 2 and 3, which causes the function to execute and return 5. This return value is assigned to the variable result, which is then printed to the console.

So, in summary, a function is a block of code that performs a specific task, while a function call is an instruction to execute a function at a particular point in a program with input arguments (if any) and a return value (if any).

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

In Python, there is only one global scope per program. This means that any variables or functions defined outside of any function or class are in the global scope and can be accessed from anywhere in the program.

On the other hand, local scopes are created whenever a function is called. This means that each function call has its own local scope, which is separate from the global scope and from the local scopes of any other function calls. Variables defined inside a function are in the local scope of that function and can only be accessed within that function.

To summarize:

There is only one global scope per Python program.

Local scopes are created whenever a function is called, and each function call has its own local scope.

Variables defined outside of any function or class are in the global scope and can be accessed from anywhere in the program.

Variables defined inside a function are in the local scope of that function and can only be accessed within that function.

It's important to keep in mind the differences between global and local scopes when working with variables in Python to avoid any unexpected behavior or bugs in your code.

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

When a function call returns in Python, the local scope of that function is destroyed, and any variables defined within the local scope are also destroyed. This means that the variables defined in a local scope only exist for the duration of the function call, and they cannot be accessed from outside the function.

For example, consider the following code:

def my\_function():

x = 5

print(x)

my\_function()

print(x) # Raises a NameError: name 'x' is not defined

In this example, we define a function called my\_function that creates a variable x and prints its value. We then call the function, which prints the value of x (which is 5). However, when we try to print the value of x outside the function, we get a NameError because x only exists within the local scope of the function and is destroyed when the function call returns.

It's important to keep in mind that variables defined in a local scope do not affect variables in the global scope, and vice versa. If you need to access or modify a global variable from within a function, you can use the global keyword to indicate that you are referring to the global variable, as follows:

x = 0

def my\_function():

global x

x = 5

my\_function()

print(x) # Output: 5

In this example, we use the global keyword to indicate that we are referring to the global variable x, which is defined outside the function. We then call the function, which modifies the value of x to 5. When we print the value of x outside the function, we get 5 because the function modified the global variable.

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

In Python, a return value is the value that a function returns when it is called. A function can return any type of value, including numbers, strings, lists, dictionaries, and even other functions. The return value of a function is typically assigned to a variable or used as part of an expression.

For example, consider the following code:

python

Copy code

def add\_numbers(x, y):

return x + y

result = add\_numbers(2, 3)

print(result)

In this example, we define a function called add\_numbers that takes two input parameters and returns their sum using the return statement. We then call this function with arguments 2 and 3, which causes the function to execute and return 5. This return value is assigned to the variable result, which is then printed to the console.

It is also possible to have a return value in an expression, which means that the return value of a function can be used as part of a larger expression. For example:

def multiply\_numbers(x, y):

return x \* y

result = add\_numbers(2, multiply\_numbers(3, 4))

print(result)

In this example, we define a function called multiply\_numbers that takes two input parameters and returns their product using the return statement. We then call the add\_numbers function with arguments 2 and the result of calling multiply\_numbers with arguments 3 and 4. This causes multiply\_numbers to execute and return 12, which is then used as the second argument to add\_numbers. The add\_numbers function then returns the sum of 2 and 12, which is 14. This return value is assgned to the variable result, which is then printed to the console.

In summary, a return value is the value that a function returns when it is called, and it can be used as part of an expression to perform more complex calculations.

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

If a function in Python does not have a return statement, then the return value of a call to that function is None. None is a built-in constant in Python that represents the absence of a value, and it is often used to indicate that a function has no return value.

For example, consider the following code:

def say\_hello(name):

print("Hello, " + name + "!")

result = say\_hello("Alice")

print(result)

In this example, we define a function called say\_hello that takes a name parameter and prints a greeting to the console. However, the function does not have a return statement. We then call the function with the argument "Alice" and assign the return value to the variable result. When we print result, we get None because the say\_hello function does not return a value.

It's important to keep in mind that functions without a return statement can still have side effects, such as printing output to the console or modifying global variables. However, if you need to use the result of a function in an expression or assign it to a variable, you should make sure to include a return statement that returns the appropriate value.

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

In Python, if you want to access a global variable from within a function, you can use the global keyword to indicate that the variable should be treated as a global variable. This allows you to modify the value of the global variable from within the function.

Here's an example:

x = 10

def my\_function():

global x

x = 20

my\_function()

print(x)

In this example, we define a global variable x with a value of 10. We then define a function called my\_function that uses the global keyword to indicate that it should use the global variable x. Inside the function, we modify the value of x to 20. When we call the function and then print the value of x, we get 20, because the function modified the global variable.

It's important to use the global keyword with caution, as modifying global variables from within functions can make it harder to reason about your code and can lead to bugs. It's generally better to use function parameters to pass data into a function, and to return values from the function to communicate results back to the caller. However, in some cases, using global variables can be a useful tool, especially for simple scripts or prototyping.

10. What is the data type of None?

In Python, None is a built-in constant that represents the absence of a value. It is often used to indicate that a function has no return value or to represent missing or undefined data.

The data type of None is NoneType. NoneType is a built-in type in Python that has only one possible value, which is None. You can check the data type of None using the type() function, like this:

x = None

print(type(x)) # Output: <class 'NoneType'>

This code creates a variable x and assigns it the value None. We then use the type() function to check the data type of x, which is NoneType.

11. What does the sentence import areallyourpetsnamederic do?

The sentence import areallyourpetsnamederic is not a valid Python import statement and would result in a ModuleNotFoundError.

In Python, the import statement is used to import modules, which are collections of functions, classes, and variables that can be used in a Python program. When you import a module, you can use its functions, classes, and variables in your program by referencing them with the module name.

For example, if you have a module called math that provides mathematical functions like sqrt() and sin(), you can import it and use its functions like this:

import math

x = math.sqrt(16)

print(x) # Output: 4.0

In this code, we import the math module using the import statement. We then use the sqrt() function from the math module to compute the square root of 16 and assign the result to the variable x. Finally, we print the value of x.

If you try to import a module that does not exist or is not installed on your system, you will get a ModuleNotFoundError.

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

If you imported a module called spam that contains a function called bacon(), you would call it using the following syntax:

import spam

spam.bacon()

In this code, we import the spam module using the import statement, and then call the bacon() function from the spam module using the dot notation. The dot notation allows you to access the functions, classes, and variables defined in the module.

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

There are several ways to prevent a program from crashing if it encounters an error:

Use try-except blocks: You can use try-except blocks to catch exceptions and handle errors gracefully. The code that might raise an exception goes in the try block, and the code that handles the exception goes in the except block. For example:

try:

# Code that might raise an exception

except SomeException:

# Code that handles the exception

Use if statements to check for errors: You can use if statements to check for errors and handle them accordingly. For example:

if x == 0:

# Handle the error

else:

# Continue with the program

Use assertions: You can use assertions to check for certain conditions that must be true and raise an exception if they are not. This can help you catch errors early in development. For example:

assert x != 0, "x should not be zero"

Use logging: You can use the Python logging module to log error messages and other information. This can help you debug your program and identify errors. For example:

import logging

logging.basicConfig(filename='example.log', level=logging.DEBUG)

try:

# Code that might raise an exception

except SomeException as e:

logging.error(e)

By using these techniques, you can make your programs more robust and handle errors in a more graceful way.

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

The purpose of the try clause in Python is to enclose a block of code that might raise an exception or error during its execution. The try clause lets you test a block of code for errors or exceptions without stopping the program's execution. If an exception is raised during the execution of the try block, the control is transferred to the corresponding except block.

The purpose of the except clause in Python is to define the block of code that should be executed if an exception is raised in the corresponding try block. The except block can specify the type of exception that it is handling, and it can define how to handle that exception. If an exception occurs during the execution of the try block, the corresponding except block is executed, and the program continues to run.

Here is an example that demonstrates the use of the try and except clauses:

try:

x = int(input("Enter a number: "))

y = int(input("Enter another number: "))

result = x / y

print("The result is:", result)

except ZeroDivisionError:

print("Error: You cannot divide by zero!")

In this code, we use the try clause to enclose the code that might raise an exception during its execution. We then use the except clause to specify how to handle the ZeroDivisionError exception if it is raised. If the user enters 0 for y, the ZeroDivisionError exception will be raised, and the corresponding except block will be executed. The program will not crash, and the user will receive an error message instead.