Python Class 11 Series

Functions - Chapter: 7

Saroj Kumar Jha

Saroj Codes

Please subscribe

INTRODUCTION

In Python, a function is a reusable block of code that performs a specific task or set of tasks. It allows you to break down your code into smaller, more manageable pieces, making it easier to read, understand, and maintain. Functions enhance code reusability and modularity by allowing you to call the same block of code multiple times from different parts of your program.

To define a function in Python, you use the def keyword, followed by the function name and a set of parentheses containing optional parameters. The basic syntax for creating a function looks like this:

```
def function_name(parameter1, parameter2, ...):
    # Function body (code block)
    # Perform tasks here
    return result  # Optional return statement
```

USER DEFINED FUNCTIONS

User-defined functions, as the name suggests, are functions created by users or programmers to perform specific tasks according to their requirements. These functions provide a way to modularize code and make it more organized, efficient, and reusable.

Creating a user-defined function in Python involves defining the function using the def keyword and specifying its name, parameters, and code block. Let's go through the steps of creating and using a user-defined function:

Function Definition:

To create a user-defined function, use the def keyword followed by the function name and a set of parentheses containing optional parameters.

```
def function_name(parameter1, parameter2, ...):
    # Function body (code block)
    # Perform tasks here
    return result  # Optional return statement
```

USER DEFINED FUNCTIONS

Function Parameters:

Parameters (also known as arguments) are values that the function accepts as inputs. You can have zero or more parameters. If there are no parameters, the parentheses are left empty. Parameters act as placeholders for the values you'll pass when calling the function.

Function Body:

The function body contains the code that the function executes when called. This code can consist of one or more statements that perform specific tasks. The code inside the function is indented to separate it from the rest of the code.

Return Statement (Optional):

The return statement is optional in a function. If included, it allows the function to return a value as the result of its execution. If there is no return statement, the function returns None by default.

USER DEFINED FUNCTIONS

Here's a simple example of a user-defined function that adds two numbers:

```
def add_numbers(num1, num2):
    sum_result = num1 + num2
    return sum_result

# Call the function and print the result
result = add_numbers(5, 7)
print(result)  # Output: 12
```

In this example, add_numbers is a user-defined function that takes two parameters (num1 and num2) and returns their sum.

User-defined functions can be used to encapsulate blocks of code, making it easier to read and maintain your code. As your programs become more complex, using functions helps you organize your code into logical units, promoting code reusability and making debugging and testing more manageable.

Arguments and Parameters

In programming, the terms "arguments" and "parameters" are often used in the context of functions. While they are related, they have distinct meanings:

Parameters:

Parameters are variables or placeholders that you define in the function definition. They act as placeholders for the values that will be passed into the function when it is called. Parameters are specified in the function's parentheses.

Example:

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

In this example, x and y are parameters of the function add_numbers. They are defined in the function's signature to receive two values when the function is called.

Arguments and Parameters

Arguments:

Arguments are the actual values or expressions that are passed into a function when it is called. These values are assigned to the corresponding parameters in the function definition.

Example:

result = add_numbers(5, 7)

In this example, 5 and 7 are arguments. When the function add_numbers is called with these arguments, the values 5 and 7 will be assigned to the parameters x and y, respectively.

To summarize, parameters are variables declared in the function definition, while arguments are the values passed to the function when it is called. When you call a function and provide values as arguments, those arguments are matched to the function's parameters in the order they appear. It's essential to ensure that the number and order of arguments match the number and order of parameters in the function definition to avoid errors.

Default Parameter

In Python, a default parameter is a parameter in a function definition that has a predefined default value. When you call the function and do not provide a value for a parameter with a default value, the default value is used instead.

You define default parameters in the function signature by assigning a default value to the parameter. Here's the syntax for defining a function with default parameters:

```
def function_name(parameter1=default_value1, parameter2=default_value2, ...):
    # Function body
    # Perform tasks here
    return result
```

In the above function definition, parameter1, parameter2, etc., are the parameters, and default_value1, default_value2, etc., are the default values assigned to these parameters.

Default Parameter

Let's see an example of a function with default parameters:

```
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"

# Call the function with both parameters provided
result1 = greet("John", "Hi")
print(result1)  # Output: "Hi, John!"

# Call the function with only the first parameter provided (uses default for the second parameter)
result2 = greet("Alice")
print(result2)  # Output: "Hello, Alice!"
```

Functions Returning Value

In Python, functions can return values using the return statement. The return statement is used to exit the function and send a result back to the caller. When a function returns a value, it can be assigned to a variable or used directly in expressions.

Here's the basic syntax of a function that returns a value:

```
def function_name(parameters):
    # Function body
    # Perform tasks here
    return result  # Return a value
```

When the return statement is encountered in the function, the function execution stops, and the value specified in the return statement is sent back as the result. If the return statement is omitted, the function returns None by default.

Let's see an example of a function that calculates the area of a rectangle and returns the result:

Functions Returning Value

```
def calculate_rectangle_area(length, width):
    area = length * width
    return area

# Call the function and store the result in a variable
result_area = calculate_rectangle_area(5, 7)
print(result_area)  # Output: 35
```

In this example, the calculate_rectangle_area function takes two parameters (length and width) and calculates the area of the rectangle using the formula length * width. The result is then returned using the return statement.

You can use the return value of a function in various ways, such as printing it, storing it in a variable, passing it to other functions, or using it in expressions.

Functions Returning Value

```
def square(number):
    return number * number
result = square(3)
print(result)  # Output: 9
result_sum = square(2) + square(4)
print(result_sum)  # Output: 20
```

A function can also return multiple values using tuples or other data structures. In such cases, you can unpack the returned values when calling the function.

```
def calculate_rectangle_properties(length, width):
    area = length * width
    perimeter = 2 * (length + width)
    return area, perimeter
result_area, result_perimeter = calculate_rectangle_properties(5, 7)
print(result_area)  # Output: 35
print(result_perimeter)  # Output: 24
```

Returning values from functions allows you to encapsulate and reuse functionality throughout your code and make your code more organized and efficient.

Flow of Execution

The "flow of execution" refers to the order in which statements in a program are executed by the computer. When you run a program, the computer starts executing statements from the top of the script and continues sequentially, line by line, until it reaches the end of the program or encounters a control flow statement (such as loops, conditionals, or function calls) that alters the normal sequence of execution.

Let's understand the flow of execution in a simple Python program:

```
# A simple Python program
def greet(name):
    print(f"Hello, {name}!")
def add_numbers(a, b):
    return a + b
print("Start of the program")
greet("Alice")
x = 5
y = 10
result = add_numbers(x, y)
print(f"The sum of {x} and {y} is: {result}")
print("End of the program")
```

Flow of Execution

Flow of execution for the above program:

1. The program starts executing from the top. It encounters the greet function definition but doesn't execute the function body yet.

2. Next, it defines the add_numbers function but doesn't execute the function body yet.

3. The print("Start of the program") statement is executed, and it prints "Start of the program" to the console.

4. The greet("Alice") statement is executed, and it calls the greet function with the argument "Alice." The flow of execution jumps to the greet function, executes its body (print(f"Hello, {name}!")), and prints "Hello, Alice!" to the console.

5. The flow of execution returns to the main program and continues with the next statement.

Flow of Execution

6. The variables x and y are assigned values 5 and 10, respectively.
7. The result = add_numbers(x, y) statement calls the add_numbers function with arguments x and y. The flow of execution jumps to the add_numbers function, executes its body (return a + b), and returns the result (15).
8. The flow of execution returns to the main program and continues with the next statement.
9. The program prints "The sum of 5 and 10 is: 15" to the console.
10. Finally, the print("End of the program") statement is executed, and it prints "End of the program" to the console.

The flow of execution is critical for understanding how a program behaves, how variables change their values, and how functions interact with each other during runtime. Understanding the flow of execution helps in identifying and debugging issues in the code and designing effective algorithms and control structures.

SCOPE OF A VARIABLE

The "scope" of a variable in Python refers to the region of the code where the variable is accessible and can be referenced. In other words, it defines the context in which a variable can be used and where it holds a valid value. The scope of a variable is determined by where it is defined in the code.

In Python, there are mainly three types of variable scope:

Global Scope:

Variables defined outside any function or block have a global scope. They can be accessed from any part of the code, including inside functions. To declare a variable with global scope, you simply define it outside of any function.

Example:

```
global_var = 10  # Global variable
def some_function():
    print(global_var)  # Accessing the global variable
some_function()  # Output: 10
```

SCOPE OF A VARIABLE

Local Scope:

Variables defined inside a function have a local scope. They are only accessible within that specific function and are not visible to the rest of the code outside the function.

Example:

```
def some_function():
    local_var = 20  # Local variable
    print(local_var)
some_function()  # Output: 20
# print(local_var)  # This would raise an error because local_var is not accessible here.
```

Enclosing Scope (Nested Functions):

When you have nested functions (a function inside another function), the inner function can access variables from the outer (enclosing) function's scope.

SCOPE OF A VARIABLE

Example:

```
def outer_function():
    outer_var = 30

    def inner_function():
        print(outer_var)  # Accessing the variable from the enclosing function

    inner_function()  # Output: 30

outer_function()
```

In this example, the inner_function can access the variable outer_var, which is defined in the enclosing outer_function.

SCOPE OF A VARIABLE

Python follows the "LEGB" rule for variable resolution:

Local scope: Variables defined within the current function.
Enclosing scope: Variables defined in any enclosing functions (if applicable).
Global scope: Variables defined at the top level of the module.
Built-in scope: Predefined names like print, len, etc. that are available in all modules.
If a variable is not found in the current scope, Python will search in the enclosing scopes, global scope, and built-in scope, in that order.

Understanding variable scope is essential to avoid naming conflicts, manage data efficiently, and write clean and maintainable code. When using global variables inside functions, you may need to use the global keyword to indicate that you want to modify the global variable, not create a new local variable with the same name.

Built-in functions

Built-in functions in Python are pre-defined functions that are available as part of the Python programming language. These functions are always accessible and do not require any additional imports or installations. They serve as a fundamental part of the Python language and provide essential functionalities for various tasks.

Here are some commonly used built-in functions in Python:

1. print(): Used to display output on the screen.
2. input(): Allows the user to input data from the keyboard.
3. len(): Returns the length (number of items) of an object like a string, list, tuple, etc.
4. range(): Generates a sequence of numbers.
5. int(), float(), str(), list(), tuple(), dict(), set(): Functions to convert data types to integers, floats, strings, lists, tuples, dictionaries, and sets, respectively.
6. sorted(): Sorts a sequence (list, tuple, etc.) and returns a new sorted list.

Built-in functions

7. max(), min(): Returns the maximum or minimum value from a sequence.
8. sum(): Calculates the sum of all elements in a list or tuple.
9. abs(): Returns the absolute value of a number.
10. round(): Rounds a number to a specified number of decimal places.
11. enumerate(): Adds a counter to an iterable and returns it as an enumerate object.
12. zip(): Combines multiple iterables into tuples of corresponding elements.
13. any(), all(): Check if any or all elements in an iterable are true.
14. type(): Returns the type of an object.
15. dir(): Returns a list of names in the current local scope or the attributes of an object.
16. id(): Returns the unique identity of an object (memory address).

These are just a few examples of the built-in functions available in Python. The Python Standard Library provides a wide range of built-in functions for various purposes. Additionally, Python has many modules and packages that extend the functionality of the language and offer more specialized functions for specific tasks.

You can use these built-in functions directly in your Python code without any additional setup or import statements, making them readily accessible and convenient to use.

Built-in functions

print(): Used to display output on the screen.

```
print("Hello, World!")  # Output: Hello, World!
```

len(): Returns the length (number of items) of an object like a string, list, tuple, etc.

```
text = "Python"
length = len(text)
print(length)  # Output: 6
```

input(): Allows the user to input data from the keyboard.

```
name = input("Enter your name: ")
print("Hello, " + name + "!")  # Output: Hello, John! (if the user enters "John")
```

Built-in functions

int(), float(), str(), list(), tuple(), dict(), set(): Functions to convert data types to integers, floats, strings, lists, tuples, dictionaries, and sets, respectively.

```
number_str = "123"
number_int = int(number_str)
print(number_int)  # Output: 123
pi_float = 3.14159
pi_str = str(pi_float)
print(pi_str)  # Output: '3.14159'
values_list = [1, 2, 3]
values_tuple = tuple(values_list)
print(values_tuple)  # Output: (1, 2, 3)
```

range(): Generates a sequence of numbers.

```
numbers = list(range(5))
print(numbers)  # Output: [0, 1, 2, 3, 4]
```

Built-in functions

sum(): Calculates the sum of all elements in a list or tuple.

```
numbers = [1, 2, 3, 4, 5]
total_sum = sum(numbers)
print(total_sum)  # Output: 15
```

max(), min(): Returns the maximum or minimum value from a sequence.

```
numbers = [5, 10, 2, 25, 7]
max_value = max(numbers)
min_value = min(numbers)
print(max_value)  # Output: 25
print(min_value)  # Output: 2
```

sorted(): Sorts a sequence (list, tuple, etc.) and returns a new sorted list.

```
numbers = [5, 2, 10, 7, 1]
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # Output: [1, 2, 5, 7, 10]
```

Built-in functions

abs(): Returns the absolute value of a number.

```
num = -10
abs_num = abs(num)
print(abs_num)  # Output: 10
```

enumerate(): Adds a counter to an iterable and returns it as an enumerate object.

```
fruits = ['apple', 'banana', 'orange']
for index, fruit in enumerate(fruits):
    print(index, fruit)
# Output:
# 0 apple
# 1 banana
# 2 orange
```

any(), all(): Check if any or all elements in an iterable are true.

```
numbers1 = [0, 1, 2, 3, 4]
numbers2 = [1, 2, 3, 4, 5]
# Using 'any()' to check if any element is true (non-zero)
result_any = any(numbers1)
```

Built-in functions

any(), all(): Check if any or all elements in an iterable are true.

```
numbers1 = [0, 1, 2, 3, 4]
numbers2 = [1, 2, 3, 4, 5]
    # Using 'any()' to check if any element is true (non-zero)
result_any = any(numbers1)
print(result_any)  # Output: True (At least one non-zero element)
    # Using 'all()' to check if all elements are true (non-zero)
result_all = all(numbers2)
print(result_all)  # Output: True (All elements are non-zero)
```

zip(): Combines multiple iterables into tuples of corresponding elements.

```
names = ['Alice', 'Bob', 'Charlie']
scores = [85, 92, 78]
# Using 'zip()' to combine 'names' and 'scores'
name_score_pairs = list(zip(names, scores))
print(name_score_pairs)  # Output: [('Alice', 85), ('Bob', 92), ('Charlie', 78)]
```

Built-in functions

type(): Returns the type of an object.

```
number = 42
text = "Hello"
my_list = [1, 2, 3]
print(type(number))  # Output: <class 'int'>
print(type(text))    # Output: <class 'str'>
print(type(my_list)) # Output: <class 'list'>
```

sorted() : with custom key function:

Sorting a list of strings based on their length using a custom key function.

```
fruits = ['apple', 'banana', 'orange', 'kiwi’]
 # Sorting based on the length of the strings using a lambda function as the key
sorted_fruits = sorted(fruits, key=lambda x: len(x))
print(sorted_fruits)  # Output: ['kiwi', 'apple', 'banana', 'orange']
```

Built-in functions

id(): Returns the unique identity of an object (memory address).

```
x = 42
y = x
print(id(x))  # Output: Memory address of x
print(id(y))  # Output: Same memory address as x
```

round(): Rounds a number to a specified number of decimal places.

```
pi = 3.14159
# Rounding 'pi' to 2 decimal places
rounded_pi = round(pi, 2)
print(rounded_pi)  # Output: 3.14
```

Built-in functions

dir(): Returns a list of names in the current local scope or the attributes of an object.

```
# Using 'dir()' to list names in the current scope
local_names = dir()
print(local_names)  # Output: List of names in the current scope
# Using 'dir()' to list attributes of a list object
numbers = [1, 2, 3]
list_attributes = dir(numbers)
print(list_attributes)  # Output: List of attributes of the 'numbers' list
```

Module

In Python, a module is a file containing Python code that defines functions, classes, and variables. Modules serve as a way to organize and reuse code by grouping related functionality together. They allow you to break your program into smaller, more manageable pieces, making it easier to read, understand, and maintain.

A module can contain Python code, including variable and function definitions, class definitions, and even other modules. To use the code from a module, you need to import it into your program using the import statement.

Built-in Modules

In Python, the term "built-in modules" refers to a set of standard modules that are automatically available in the Python environment without requiring any additional installation or setup. These modules are part of the Python Standard Library, which comes with the Python interpreter, and they provide a wide range of functionalities for various tasks, such as mathematical operations, file handling, working with dates and times, random number generation, and more.

Here are some commonly used built-in modules in Python:

math: Provides mathematical functions and constants.
import math
random: Generates random numbers and sequences.
import random
datetime: Manipulates dates and times.
import datetime

Built-in Modules

os: Provides functions for interacting with the operating system.

import os

json: Encodes and decodes JSON data.

import json

sys: Provides access to some variables used or maintained by the Python interpreter and functions that interact with the interpreter.

import sys

time: Provides various time-related functions.

import time

statistics : module in Python is a built-in module that provides functions for statistical calculations. It offers a set of statistical operations that allow you to compute mean, median, variance, standard deviation, and more for a given set of numeric data.

math

The math module is a built-in module in Python that provides mathematical functions and constants. It is one of the most commonly used modules for performing various mathematical operations. To use the math module, you need to import it into your Python script using the import statement.

Here are some of the commonly used functions and constants from the math module:

math.sqrt(): Calculates the square root of a number.

```
import math
result = math.sqrt(25)
print(result)  # Output: 5.0
```

math.pow(): Calculates the power of a number.

```
import math
result = math.pow(2, 3)  # Equivalent to 2 ** 3
print(result)  # Output:
```

math

math.sin(), math.cos(), math.tan(): Calculates trigonometric functions in radians.

```
import math
angle = math.radians(30)  # Convert 30 degrees to radians
sin_value = math.sin(angle)
cos_value = math.cos(angle)
tan_value = math.tan(angle)
print(sin_value)  # Output: 0.4999999999999994 (approximately 0.5)
print(cos_value)  # Output: 0.8660254037844387 (approximately 0.866)
print(tan_value)  # Output: 0.5773502691896257 (approximately 0.577)
```

math.pi: A constant representing the value of π (pi).

```
import math
print(math.pi)  # Output: 3.141592653589793
```

math.e: A constant representing the base of the natural logarithm (e).

```
import math
print(math.e)  # Output: 2.718281828459045
```

math

math.ceil(), math.floor(), and math.trunc(): Rounding functions.

```
import math
print(math.ceil(3.5))   # Output: 4
print(math.floor(3.5))  # Output: 3
print(math.trunc(3.5))  # Output: 3
```

math.factorial(): Calculates the factorial of a number.

```
import math
result = math.factorial(5)  # Equivalent to 5 * 4 * 3 * 2 * 1
print(result)  # Output: 120
```

These are just a few examples of the functions and constants provided by the math module. The math module offers many more mathematical functions that can be very useful for a wide range of applications, including scientific calculations, geometry, statistics, and more.

random

The random module is another built-in module in Python that provides functions for generating random numbers, sequences, and making random choices. It is useful for scenarios where you need to introduce randomness or unpredictability in your programs, such as in games, simulations, and statistical simulations.

To use the random module, you need to import it into your Python script using the import statement.

Here are some of the commonly used functions from the random module:

random.random(): Generates a random floating-point number between 0 and 1 (exclusive).

```
import random
random_number = random.random()
print(random_number)  # Output: A random float between 0 and 1 (e.g., 0.54321)
```

random.randint(): Generates a random integer within a specified range (inclusive).

```
import random
random_int = random.randint(1, 10)  # Random integer between 1 and 10 (inclusive
print(random_int)  # Output: A random integer between 1 and 10 (e.g., 5)
```

random

random.choice(): Picks a random element from a sequence (e.g., list, tuple, string).

```
import random
colors = ['red', 'green', 'blue', 'yellow']
random_color = random.choice(colors)
print(random_color)  # Output: A random color from the list (e.g., 'blue')
```

random.shuffle(): Randomly shuffles the elements of a list.

```
import random
numbers = [1, 2, 3, 4, 5]
random.shuffle(numbers)
print(numbers)  # Output: A shuffled version of the list (e.g., [3, 1, 4, 5, 2])
```

random.sample(): Generates a random sample (subset) from a sequence without replacement.

```
import random
numbers = [1, 2, 3, 4, 5]
random_sample = random.sample(numbers, 3)  # Random sample of size 3
print(random_sample)  # Output: A random subset of the list (e.g., [2, 4, 1])
```

random

random.uniform(): Generates a random floating-point number within a specified range.

```
import random
random_float = random.uniform(1.0, 5.0)  # Random float between 1.0 and 5.0 (inclusive)
print(random_float)  # Output: A random float between 1.0 and 5.0 (e.g., 3.4567)
```

random.randrange() is a function from the random module in Python that generates a random integer within a specified range. It allows you to choose a random integer from a start value (inclusive) up to, but not including, an end value. You can also specify a step size to increment the range.

The syntax of random.randrange() is as follows:

```
random.randrange(start, stop[, step])
```

Parameters:

start: The starting value of the range (inclusive).

stop: The stopping value of the range (exclusive).

step (optional): The step size between numbers. The default value is 1.

random

Here are some examples of how to use random.randrange():

Generating a random integer from 0 to 9:

```
import random
random_number = random.randrange(10)
print(random_number)  # Output: A random integer from 0 to 9 (e.g., 5)
```

Generating a random even number from 2 to 10:

```
import random
random_even = random.randrange(2, 11, 2)
print(random_even)  # Output: A random even number from 2 to 10 (e.g., 6)
```

Generating a random multiple of 5 from 10 to 100:

```
import random
random_multiple_of_5 = random.randrange(10, 101, 5)
print(random_multiple_of_5)  # Output: A random multiple of 5 from 10 to 100 (e.g., 65)
```

random

Shuffling a list randomly using random.randrange():

```
import random
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print(my_list)  # Output: A shuffled version of the list (e.g., [3, 5, 2, 1, 4])
```

statistics

To use the statistics module, you need to import it into your Python script. Here are some of the commonly used functions from the statistics module:

mean(): Calculates the arithmetic mean (average) of a list of numbers.

```
import statistics
data = [10, 20, 30, 40, 50]
mean_value = statistics.mean(data)
print(mean_value)  # Output: 30
```

median(): Calculates the median (middle value) of a list of numbers.

```
import statistics
data = [10, 20, 30, 40, 50]
median_value = statistics.median(data)
print(median_value)  # Output: 30
```

mode(): Calculates the mode (most common value) of a list of numbers.

```
import statistics
data = [10, 20, 30, 40, 30, 50]
mode_value = statistics.mode(data)
print(mode_value)  # Output: 30
```

statistics

variance(): Calculates the variance of a sample of numbers.

```
import statistics
data = [10, 20, 30, 40, 50]
variance_value = statistics.variance(data)
print(variance_value)  # Output: 250
```

stdev(): Calculates the standard deviation of a sample of numbers.

```
import statistics
data = [10, 20, 30, 40, 50]
stdev_value = statistics.stdev(data)
print(stdev_value)  # Output: 15.811388300841896
```

median_low() , median_high(): Calculate the low and high median values, respectively, of a list of numbers.

```
import statistics
data = [10, 20, 30, 40, 50]
median_low_value = statistics.median_low(data)
median_high_value = statistics.median_high(data)
print(median_low_value)  # Output: 30
print(median_high_value) # Output: 30
```

From Statement

In Python, when importing a module, you can use the from statement to import specific names (functions, classes, or variables) from the module directly into your code's namespace. This approach allows you to use those names without explicitly referencing the module name.

The syntax for using the from statement is as follows:

from module_name import name1, name2, ...

Where:

module_name: The name of the module you want to import from.

name1, name2, ...: The specific names (functions, classes, or variables) you want to import from the module.

EXERCISE

Q. Observe the following programs carefully, and identify the error:

a) def create (text, freq):
 for i in range (1, freq):
 print text
create(5) #function call

b) from math import sqrt,ceil
def calc():
 print cos(0)
calc() #function call

c) mynum = 9
def add9():
 mynum = mynum + 9
 print mynum
add9() #function call

d) def findValue(vall = 1.1, val2, val3):
 final = (val2 + val3)/ vall
 print(final)
findvalue() #function call

e) def greet():
 return("Good morning")
greet() = message #function call

EXERCISE

a) Error: In the function call create(5), it is missing the second argument freq. The create() function expects two arguments (text and freq), but only one argument (5) is provided.

b) Error: In the function calc(), cos() is used without a module prefix (math). The correct usage should be math.cos(0) because we only imported sqrt and ceil from the math module, not cos.

c) Error: The add9() function tries to access the global variable mynum and modify its value. However, it cannot directly modify the global variable because it is trying to perform both read and write operations on mynum. To modify the global variable within a function, you need to use the global keyword to declare it as a global variable.

EXERCISE

d) Error: In the function definition, findValue(vall=1.1, val2, val3), the default value (1.1) is provided for the first parameter vall, but no default value is provided for val2 and val3. Python requires that parameters with default values should come after parameters without default values. The correct function definition should be:

e) Error: The line greet() = message is incorrect. You cannot assign a value to the result of a function call. Instead, you should store the result of the function call in a variable:

Corrected versions of the programs:

a)

```
def create(text, freq):
    for i in range(1, freq):
        print(text)
create("Hello", 5)  # function call
```

EXERCISE

b)

```
from math import sqrt, cos
def calc():
    print(cos(0))
calc()  # function call
```

c)

```
mynum = 9
def add9():
    global mynum
    mynum = mynum + 9
    print(mynum)
add9()  # function call
```

EXERCISE

d)
```
def findValue(val2, val3, vall=1.1):
    final = (val2 + val3) / vall
    print(final)
findValue(2, 3)  # function call
```

e)
```
def greet():
    return "Good morning"
message = greet()  # function call
```

EXERCISE

Q. To secure your account, whether it be an email, online bank account or any other account, it is important that we use authentication. Use your programming expertise to create a program using user defined function named login that accepts userid and password as parameters (login(uid,pwd)) that displays a message “account blocked” in case of three wrong attempts. The login is successful if the user enters user ID as "ADMIN" and password as "St0rE@1". On successful login, display a message “login successful”

```
def login(uid, pwd):
    attempts = 0
    max_attempts = 3
    correct_uid = "ADMIN"
    correct_pwd = "St0rE@1"
    while attempts < max_attempts:
        if uid == correct_uid and pwd == correct_pwd:
            print("Login successful")
            return
        else:
            attempts += 1
            print("Invalid credentials. Please try again.")
    print("Account blocked")
# Get user input for userid and password
user_id = input("Enter User ID: ")
password = input("Enter Password: ")
# Call the login function with user input as arguments
login(user_id, password)
```

EXERCISE

Q. XYZ store plans to give festival discount to its customers. The store management has decided to give discount on the following criteria: Shopping Amount Discount Offered

>=500 and <1000 5%

>=1000 and <2000 8%

>=2000 10%

An additional discount of 5% is given to customers who are the members of the store. Create a program using user defined function that accepts the shopping amount as a parameter and calculates discount and net amount payable on the basis of the following conditions:

Net Payable Amount = Total Shopping Amount – Discount.

EXERCISE

```
def calculate_discount(shopping_amount, is_member):
    if shopping_amount >= 2000:
        discount_percentage = 10
    elif shopping_amount >= 1000:
        discount_percentage = 8
    elif shopping_amount >= 500:
        discount_percentage = 5
    else:
        discount_percentage = 0

    if is_member:
        discount_percentage += 5

    discount = (shopping_amount * discount_percentage) / 100
    net_payable_amount = shopping_amount - discount
```

EXERCISE

```
    return discount, net_payable_amount

# Get user input for shopping amount and membership status
shopping_amount = float(input("Enter Shopping Amount: "))
is_member = input("Are you a member of the store? (yes/no): ").lower() == "yes"

# Call the function to calculate the discount and net payable amount
discount_amount, net_amount_payable = calculate_discount(shopping_amount,
is_member)

print(f"Discount Amount: {discount_amount:.2f}")
print(f"Net Payable Amount: {net_amount_payable:.2f}")
```

EXERCISE

Q. ‘Play and learn’ strategy helps toddlers understand concepts in a fun way. Being a senior student you have taken responsibility to develop a program using user defined functions to help children master two and three-letter words using English alphabets and addition of single digit numbers. Make sure that you perform a careful analysis of the type of questions that can be included as per the age and curriculum.

```
import random

# Function to introduce two-letter words
def learn_two_letter_words():
    two_letter_words = ["at", "it", "on", "up", "go", "my", "an", "am"]

    print("Let's learn two-letter words!")
    while True:
        word = random.choice(two_letter_words)
        print("What word starts with the letter '{}'?".format(word[0]))
        user_input = input("Enter your answer (type 'exit' to stop): ").lower()

        if user_input == 'exit':
            break
```

EXERCISE

```
        if user_input == word:
            print("Correct! Good job!")
        else:
            print("Oops! Try again!")

# Function to introduce three-letter words
def learn_three_letter_words():
    three_letter_words = ["cat", "dog", "sun", "hat", "car", "pen", "bus", "cup"]

    print("Let's learn three-letter words!")
    while True:
        word = random.choice(three_letter_words)
        print("What word starts with the letter '{}'?".format(word[0]))
        user_input = input("Enter your answer (type 'exit' to stop): ").lower()

        if user_input == 'exit':
            break

        if user_input == word:
            print("Correct! You got it right!")
        else:
            print("Oops! Try again!")
```

EXERCISE

```
# Function to practice single-digit addition
def addition_game():
    print("Let's practice addition of single-digit numbers!")
    while True:
        num1 = random.randint(1, 9)
        num2 = random.randint(1, 9)
        correct_answer = num1 + num2

        print("What is {} + {}?".format(num1, num2))
        user_input = input("Enter your answer (type 'exit' to stop): ")

        if user_input == 'exit':
            break

        try:
            user_answer = int(user_input)
            if user_answer == correct_answer:
                print("Great! You got it right!")
            else:
                print("Oops! That's not the correct answer.")
        except ValueError:
            print("Invalid input. Please enter a number.")
```

EXERCISE

```
# Main program to call the functions
def main():
    print("Welcome to the Play and Learn program!")
    while True:
        print("\nSelect an option:")
        print("1. Learn two-letter words")
        print("2. Learn three-letter words")
        print("3. Practice addition of single-digit numbers")
        print("4. Exit")
        choice = input("Enter your choice: ")
        if choice == '1':
            learn_two_letter_words()
        elif choice == '2':
            learn_three_letter_words()
        elif choice == '3':
            addition_game()
        elif choice == '4':
            print("Thank you for playing! Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
if __name__ == "__main__":
    main()
```

EXERCISE

Q. Take a look at the series below:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55...

To form the pattern, start by writing 1 and 1. Add them together to get 2. Add the last two numbers: 1+2 = 3.Continue adding the previous two numbers to find the next number in the series. These numbers make up the famed Fibonacci sequence: previous two numbers are added to get the immediate new number.

```
def fibonacci_sequence(n):
    fib_series = [1, 1]
    while len(fib_series) < n:
        next_num = fib_series[-1] + fib_series[-2]
        fib_series.append(next_num)
    return fib_series
# Get the desired length of the Fibonacci sequence from the user
length = int(input("Enter the length of the Fibonacci sequence: "))
# Call the function to generate the Fibonacci sequence
sequence = fibonacci_sequence(length)
# Display the generated Fibonacci sequence
print("Fibonacci Sequence:")
print(sequence)
```

EXERCISE

Q. Create a menu driven program using user defined functions to implement a calculator that performs the following:

a) Basic arithmetic operations(+,-,*,/)

b) log10(x),sin(x),cos(x)

```
import math

# Function to perform basic arithmetic operations
def perform_arithmetic_operation(num1, operator, num2):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '/':
        return num1 / num2
    else:
        return None
```

EXERCISE

```
# Function to calculate log base 10 of a number
def calculate_log_base_10(num):
    return math.log10(num)

# Function to calculate sine of a number
def calculate_sine(num):
    return math.sin(num)

# Function to calculate cosine of a number
def calculate_cosine(num):
    return math.cos(num)

# Main function to display the menu and handle user inputs
def main():
    print("Calculator Menu:")
    print("a) Basic arithmetic operations (+, -, *, /)")
    print("b) log10(x), sin(x), cos(x)")
    print("Enter 'exit' to quit")

    while True:
        choice = input("Enter your choice (a/b): ").lower()
```

EXERCISE

```
if choice == 'a':
    num1 = float(input("Enter the first number: "))
    operator = input("Enter the operator (+, -, *, /): ")
    num2 = float(input("Enter the second number: "))

    result = perform_arithmetic_operation(num1, operator, num2)
    if result is not None:
        print("Result:", result)
    else:
        print("Invalid operator. Please try again.")

elif choice == 'b':
    num = float(input("Enter a number: "))
    log_result = calculate_log_base_10(num)
    sin_result = calculate_sine(num)
    cos_result = calculate_cosine(num)

    print("log10({}) = {:.4f}".format(num, log_result))
    print("sin({}) = {:.4f}".format(num, sin_result))
    print("cos({}) = {:.4f}".format(num, cos_result))
```

EXERCISE

```
        elif choice == 'exit':
            print("Goodbye!")
            break

        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

EXERCISE

Q. Write a program to check the divisibility of a number by 7 that is passed as a parameter to the user defined function.

```
def check_divisibility_by_7(number):
    if number % 7 == 0:
        return True
    else:
        return False

# Main program to get user input and check divisibility
def main():
    try:
        num = int(input("Enter a number to check divisibility by 7: "))
        if check_divisibility_by_7(num):
            print("{} is divisible by 7.".format(num))
        else:
            print("{} is not divisible by 7.".format(num))
    except ValueError:
        print("Invalid input. Please enter an integer.")

if __name__ == "__main__":
    main()
```

EXERCISE

Q. Write a program that uses a user defined function that accepts name and gender (as M for Male,F for Female) and prefixes Mr/Ms on the basis of the gender.

```
def add_prefix(name, gender):
    if gender.upper() == 'M':
        return "Mr. " + name
    elif gender.upper() == 'F':
        return "Ms. " + name
    else:
        return name

# Main program to get user input and add prefix
def main():
    name = input("Enter your name: ")
    gender = input("Enter your gender (M/F): ")

    prefixed_name = add_prefix(name, gender)
    print("Prefixed Name:", prefixed_name)

if __name__ == "__main__":
    main()
```

EXERCISE

Q. Write a program that has a user defined function to accept the coefficients of a quadratic equation in variables and calculates its determinant. For example : if the coefficients are stored in the variables a,b,c then calculate determinant as b2-4ac. Write the appropriate condition to check determinants on positive, zero and negative and output appropriate result.

```
def calculate_determinant(a, b, c):
    determinant = b**2 - 4*a*c
    return determinant
# Main program to get user input and calculate determinant
def main():
    try:
        a = float(input("Enter the coefficient 'a': "))
        b = float(input("Enter the coefficient 'b': "))
        c = float(input("Enter the coefficient 'c': "))
        determinant = calculate_determinant(a, b, c)
        if determinant > 0:
            print("Determinant is positive. Two real and distinct roots exist.")
        elif determinant == 0:
            print("Determinant is zero. Two real and equal roots exist.")
        else:
            print("Determinant is negative. No real roots exist.")
    except ValueError:
        print("Invalid input. Please enter valid numeric coefficients.")
if __name__ == "__main__":
    main()
```

EXERCISE

Q. ABC School has allotted unique token IDs from (1 to 600) to all the parents for facilitating a lucky draw on the day of their Annual day function. The winner would receive a special prize. Write a program using Python that helps to automate the task.(Hint: use random module)

```
import random
def lucky_draw():
    # Generate a random token ID between 1 and 600
    winner_token = random.randint(1, 600)
    return winner_token
def main():
    print("Welcome to the ABC School Annual Day Lucky Draw!")
    input("Press Enter to start the lucky draw...")
    # Call the lucky_draw function to get the winner's token ID
    winner_token_id = lucky_draw()

    print(f"\nThe lucky winner is: Token ID {winner_token_id}!")
    print("Congratulations to the winner!")

if __name__ == "__main__":
    main()
```

EXERCISE

Q. Write a program that implements a user defined function that accepts Principal Amount, Rate, Time, Number of Times the interest is compounded to calculate and displays compound interest. (Hint: CI=P*(1+r/n)nt)

```
def calculate_compound_interest(principal, rate, time, n):
    # Convert rate to a decimal and calculate compound interest
    rate = rate / 100
    compound_interest = principal * (1 + rate/n)**(n*time) - principal
    return compound_interest
def main():
    print("Welcome to the Compound Interest Calculator!")
    try:
        principal = float(input("Enter the Principal Amount: "))
        rate = float(input("Enter the Rate of Interest (%): "))
        time = float(input("Enter the Time Period (years): "))
        n = int(input("Enter the Number of Times Interest is Compounded per year: "))
        # Call the calculate_compound_interest function
        compound_interest = calculate_compound_interest(principal, rate, time, n)
        print(f"\nCompound Interest: {compound_interest:.2f}")
    except ValueError:
        print("Invalid input. Please enter valid numeric values.")
if __name__ == "__main__":
    main()
```

EXERCISE

Q. Write a program that has a user defined function to accept 2 numbers as parameters, if number 1 is less than number 2 then numbers are swapped and returned, i.e., number 2 is returned in place of number1 and number 1 is reformed in place of number 2, otherwise the same order is returned

```
def swap_if_greater(num1, num2):
    if num1 < num2:
        # Swap numbers
        num1, num2 = num2, num1
    return num1, num2
def main():
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        # Call the swap_if_greater function
        result_num1, result_num2 = swap_if_greater(num1, num2)
        print("\nAfter swapping (if necessary):")
        print("Number 1:", result_num1)
        print("Number 2:", result_num2)
    except ValueError:
        print("Invalid input. Please enter valid numeric values.")
if __name__ == "__main__":
    main()
```

EXERCISE

Q. Write a program that contains user defined functions to calculate area, perimeter or surface area whichever is applicable for various shapes like square, rectangle, triangle, circle and cylinder. The user defined functions should accept the values for calculation as parameters and the calculated value should be returned. Import the module and use the appropriate functions.

```
import math

# Function to calculate the area of a square
def calculate_square_area(side_length):
    return side_length ** 2

# Function to calculate the perimeter of a square
def calculate_square_perimeter(side_length):
    return 4 * side_length

# Function to calculate the area of a rectangle
def calculate_rectangle_area(length, width):
    return length * width

# Function to calculate the perimeter of a rectangle
def calculate_rectangle_perimeter(length, width):
    return 2 * (length + width)
```

EXERCISE

```
# Function to calculate the area of a triangle
def calculate_triangle_area(base, height):
    return 0.5 * base * height

# Function to calculate the circumference of a circle
def calculate_circle_circumference(radius):
    return 2 * math.pi * radius

# Function to calculate the area of a circle
def calculate_circle_area(radius):
    return math.pi * radius ** 2

# Function to calculate the surface area of a cylinder
def calculate_cylinder_surface_area(radius, height):
    return 2 * math.pi * radius * (radius + height)

def main():
    print("Shapes Area/Perimeter/Surface Area Calculator")
```

EXERCISE

```
while True:
    print("\nSelect a shape:")
    print("1. Square")
    print("2. Rectangle")
    print("3. Triangle")
    print("4. Circle")
    print("5. Cylinder")
    print("6. Exit")

    choice = input("Enter your choice (1/2/3/4/5/6): ")

    if choice == '1':
        side_length = float(input("Enter the side length of the square: "))
        print("Area:", calculate_square_area(side_length))
        print("Perimeter:", calculate_square_perimeter(side_length))
    elif choice == '2':
        length = float(input("Enter the length of the rectangle: "))
        width = float(input("Enter the width of the rectangle: "))
        print("Area:", calculate_rectangle_area(length, width))
        print("Perimeter:", calculate_rectangle_perimeter(length, width))
```

EXERCISE

```
    elif choice == '3':
        base = float(input("Enter the base of the triangle: "))
        height = float(input("Enter the height of the triangle: "))
        print("Area:", calculate_triangle_area(base, height))
    elif choice == '4':
        radius = float(input("Enter the radius of the circle: "))
        print("Area:", calculate_circle_area(radius))
        print("Circumference:", calculate_circle_circumference(radius))
    elif choice == '5':
        radius = float(input("Enter the radius of the cylinder: "))
        height = float(input("Enter the height of the cylinder: "))
        print("Surface Area:", calculate_cylinder_surface_area(radius, height))
    elif choice == '6':
        print("Goodbye!")
        break
    else:
        print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

EXERCISE

Q. Write a program that creates a GK quiz consisting of any five questions of your choice. The questions should be displayed randomly. Create a user defined function score() to calculate the score of the quiz and another user defined function remark (scorevalue) that accepts the final score to display remarks as follows:

Marks	Remarks
5	Outstanding
4	Excellent
3	Good
2	Read more to score more
1	Needs to take interest
0	General knowledge will always help you. Take it seriously.

EXERCISE

```
import random

# Function to create the GK quiz questions
def create_quiz():
    questions = [
        {
            "question": "Which is the largest planet in our solar system?",
            "options": ["Mars", "Venus", "Jupiter", "Saturn"],
            "answer": "Jupiter"
        },
        {
            "question": "What is the capital city of France?",
            "options": ["London", "Paris", "Berlin", "Rome"],
            "answer": "Paris"
        },
        {
            "question": "Who painted the Mona Lisa?",
            "options": ["Leonardo da Vinci", "Vincent van Gogh", "Pablo Picasso", "Michelangelo"],
            "answer": "Leonardo da Vinci"
        },
```

EXERCISE

```
        {
            "question": "What is the chemical symbol for water?",
            "options": ["H2O", "CO2", "O2", "CH4"],
            "answer": "H2O"
        },
        {
            "question": "What is the tallest mountain in the world?",
            "options": ["Mount Kilimanjaro", "Mount Everest", "Mount McKinley", "Mount Fuji"],
            "answer": "Mount Everest"
        }
    ]
    return random.sample(questions, 5)

# Function to display the GK quiz questions and calculate the score
def score(questions):
    total_questions = len(questions)
    score_value = 0

    for i, question in enumerate(questions, 1):
        print(f"\nQuestion {i}: {question['question']}")
        print("Options:")
```

EXERCISE

```
        for idx, option in enumerate(question['options'], 1):
            print(f"{idx}. {option}")

        user_answer = int(input("Enter your answer (1/2/3/4): "))
        user_answer -= 1  # Adjust for 0-based indexing

        if user_answer >= 0 and user_answer < len(question['options']):
            if question['options'][user_answer] == question['answer']:
                print("Correct!")
                score_value += 1
            else:
                print("Incorrect!")

    return score_value

# Function to display remarks based on the score
def remark(score_value):
    remarks = {
        5: "Outstanding",
        4: "Excellent",
        3: "Good",
        2: "Read more to score more",
```

EXERCISE

```
        1: "Needs to take interest",
        0: "General knowledge will always help you. Take it seriously."
    }

    print("\nYour Score:", score_value)
    print("Remarks:", remarks.get(score_value, "Invalid score"))

def main():
    print("Welcome to the General Knowledge Quiz!")
    questions = create_quiz()

    # Display the quiz questions and calculate the score
    user_score = score(questions)

    # Display remarks based on the score
    remark(user_score)

if __name__ == "__main__":
    main()
```

Thank you

Saroj Kumar Jha

srojkrjha@gmail.com