Module 1 Question Bank

1) What are the different flow control statements supported in Python? Explain all the flow control statements with example code and syntax.

Solution:

Python provides several flow control statements that allow you to control the execution flow of your program. These flow control statements allow you to make decisions, iterate over sequences, and control the execution flow of your Python programs.

The main flow control statements in Python are:

1. if...elif...else statement: This statement is used for conditional execution. It allows you to perform different actions based on different conditions.

Example:

```
age = 25

if age < 18:
    print("You are underage.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

In this example, the `if...elif...else` statement checks the value of the `age` variable and prints a corresponding message based on the condition.

2. for loop: The `for` loop is used to iterate over a sequence (such as a string, list, or tuple) or other iterable objects.

Example:

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

This code iterates over each element in the `fruits` list and prints them one by one.

3. while loop: The `while` loop is used to repeatedly execute a block of code as long as a certain condition is true.

Example:

```
count = 0

while count < 5:
    print(count)
```

```
        count += 1
```

This loop prints the value of `count` and increments it by 1 until the condition `count < 5` becomes false.

4. break statement: The `break` statement is used to exit a loop prematurely. When encountered, it terminates the current loop and transfers control to the next statement after the loop.

Example:

```
numbers = [1, 2, 3, 4, 5]

for number in numbers:
    if number == 3:
        break
    print(number)
```

In this example, the loop stops executing when the value `3` is encountered, and the control flow moves to the next statement after the loop.

5. continue statement: The `continue` statement is used to skip the rest of the current iteration of a loop and move to the next iteration.

Example:

```
numbers = [1, 2, 3, 4, 5]

for number in numbers:
    if number == 3:
        continue
    print(number)
```

In this example, when the value `3` is encountered, the `continue` statement skips the rest of the current iteration. The loop continues with the next iteration, printing all the numbers except `3`.

2) Define exception handling. How exceptions are handled in python? Write a program to solve divide by zero exception.

Solution:

Exception handling is a mechanism in programming that allows you to handle and respond to errors or exceptional events that occur during the execution of a program. Exception handling provides a way to gracefully handle error situations and prevent the program from crashing.

In Python, exceptions are handled using the `try` and `except` statements. The `try` block contains the code that may potentially raise an exception, while the `except` block specifies the code to be executed if an exception occurs. The `except` block defines the type of exception it can handle. If the raised exception matches the specified type, the code inside the `except` block is executed.

Here's the basic syntax:

```
try:
    # Code that may raise an exception
    # ...
except ExceptionType:
    # Code to handle the exception
    # ...
```

Now, let's write a program to solve the divide-by-zero exception using exception handling:

```
try:
    numerator = 10
    denominator = 0
    result = numerator / denominator
    print(result)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
```

In this example, we attempt to divide `numerator` by 0. Since dividing by zero is an invalid operation and raises a `ZeroDivisionError`, we handle this exception in the `except` block. Instead of crashing the program, the exception is caught, and the error message "Error: Division by zero is not allowed" is printed.

3) Explain Local and Global Scopes in Python programs. What are local and global variables? How can you force a variable in a function to refer to the global variable?

Solution:

In Python, local and global variables are used to store values within a program. The main difference between them is their scope, which determines where the variable can be accessed and modified.

Local Variables:

Local variables are defined within a function or a block of code. They have a local scope, which means they are only accessible within the specific function or block where they are defined. Local variables cannot be accessed outside of their scope.

Example of a local variable:

```
def my_function():
    x = 10  # Local variable
    print(x)

my_function()    # Output: 10
print(x)      # Error: NameError: name 'x' is not defined
```

In this example, `x` is a local variable defined within the `my_function()` function. It can only be accessed within the function's scope. Attempting to access `x` outside of the function results in a `NameError` since it is not defined in the global scope.

Global Variables:

Global variables, as the name suggests, are defined outside of any function or block of code. They have a global scope, meaning they can be accessed and modified from anywhere in the program, including inside functions.

Example of a global variable:

```
x = 10  # Global variable
```

```
def my_function():
    print(x)

my_function()    # Output: 10
print(x)     # Output: 10
```

In this example, `x` is a global variable defined outside the function. It can be accessed and printed both inside and outside the function because of its global scope.

Forcing a Variable in a Function to Refer to the Global Variable:
We can force a variable in a function to refer to the global variable using the `global` keyword.

Example:

```
x = 10  # Global variable

def my_function():
    global x
    x = 20
    print(x)

my_function()     # Output: 20
print(x)      # Output: 20
```

In this example, the `global` keyword is used to indicate that the variable `x` inside `my_function()` refers to the global variable `x`. Therefore, when we modify `x` inside the function, it affects the global variable itself.

4) Write a program that generates a random number between 1 and 100. The program should prompt the user to guess the number and provide feedback if the guess is too high or too low. The game should continue until the user correctly guesses the number.

Solution:

```
# Program to guess a random number between 1 and 100

import random

# Generate a random secret number

secret_number = random.randint(1, 100)


# Initialize the user's guess

guess = 0


# Continue the game until the user guesses the correct number

while guess != secret_number:
    # Prompt the user to guess a number
    guess = int(input("Guess a number between 1 and 100: "))

    # Check if the guess is too high, too low, or correct
    if guess > secret_number:
        print("Too high!")
    elif guess < secret_number:
        print("Too low!")
    else:
        print("Congratulations! You guessed the correct number.")
```

Description:

This program generates a random number between 1 and 100 as the secret number. It prompts the user to guess the number and provides feedback if the guess is too high or too low. The game continues until the user correctly guesses the number.

Sample Output:

```
Guess a number between 1 and 100: 50
Too high!
Guess a number between 1 and 100: 25
Too low!
Guess a number between 1 and 100: 35
Too high!
Guess a number between 1 and 100: 30
Too low!
Guess a number between 1 and 100: 33
Too high!
Guess a number between 1 and 100: 32
Congratulations! You guessed the correct number.
```

5) Write a program that prompts the user to enter a positive integer and prints its multiplication table from 1 to 10 using a while loop.

Solution:

```
# Program to print the multiplication table of a positive integer using a while loop
```

```
number = int(input("Enter a positive integer: "))     # Prompt the user to enter a positive integer

count = 1       # Initialize the count variable to 1


while count <= 10:            # Iterate while the count is less than or equal to 10

   result = number * count                      # Calculate the result by multiplying the number and the count

   print(result)              # Print the result

   count = count + 1              # Increment the count by 1
```

Description:

This program prompts the user to enter a positive integer and then uses a while loop to print the multiplication table of that number from 1 to 10. It starts by taking user input and storing it in the `number` variable.

Sample Output:

```
Enter a positive integer: 7

7

14

21

28

35

42

49

56

63

70
```

6) Write a Python function called "calculate_factorial" that takes a positive integer as input and returns the factorial of that number.

Solution:

```
# Function to calculate the factorial of a positive integer

def calculate_factorial(n):

     factorial = 1
```

```
    for i in range(1, n + 1):
        factorial = factorial * i
    return factorial
# Main program
number = int(input("Enter a non-negative number:"))        # prompt the user to enter a non-negative number.
result = calculate_factorial(number)
print("The factorial of", number, "is:", result)
```

Description:

This program defines a function called "calculate_factorial" that calculates the factorial of a positive integer using a for loop. The user is prompted to enter a non-negative number, and the factorial of that number is calculated and printed as the output.

OUTPUT:

```
Enter a non-negative number:10
The factorial of 10 is: 3628800
```

7) Give one example each for if-else statements and while loops. Compare and contrast Python's if statement and while loop.

Solution:

Example of an if-else statement:

```
x = 5

if x > 10:
    print("x is greater than 10")
else:
    print("x is not greater than 10")
```

Example of while loop:

```
count = 0

while count < 5:
    print(count)
    count += 1
```

The if statement and while loop are both control flow structures in Python that allow you to control the execution of code based on certain conditions. The if statement is suitable for making decisions based on a condition, while the while loop is useful for performing iterative tasks as long as a condition remains true. Following is the comparison for if statement and while loop:

1) Purpose:

if statement: The if statement is used for conditional execution. It allows you to specify a condition, and if the condition evaluates to true, the code block indented under the if statement is executed.

while loop: The while loop is used for repetitive execution. It allows you to repeatedly execute a block of code as long as a specified condition remains true.

Condition:

if statement: The if statement checks a condition only once. If the condition is true, the indented code block is executed. If the condition is false, the code block is skipped.

while loop: The while loop checks a condition before each iteration. As long as the condition remains true, the code block is executed. If the condition becomes false, the loop is exited, and the program continues with the next statement.

Execution:

if statement: The code block under the if statement is executed only once, if the condition is true.

while loop: The code block under the while loop is executed repeatedly until the condition becomes false.

Control Flow:

if statement: The if statement can be followed by optional else and elif clauses to provide additional branching based on different conditions.

while loop: The while loop does not have built-in branching capabilities like else and elif. However, you can use control flow statements like break and continue to control the execution within the loop.

8) Develop a program to read the name and year of birth of a person. Display whether the person is a senior citizen or not.

Solution:

```
# Program to determine if a person is a senior citizen based on their name and year of birth


from datetime import date


def check_senior_citizen(name, year_of_birth):
    # Get the current year
    current_year = date.today().year


    # Calculate the person's age
    age = current_year - year_of_birth


    # Check if the age is 60 or above
    if age >= 60:
        return True
    else:
        return False

# Main program
name = input("Enter person's name: ")
year_of_birth = int(input("Enter year of birth: "))

# Call the function to check if the person is a senior citizen
is_senior_citizen = check_senior_citizen(name, year_of_birth)

# Display the result based on the return value of the function
if is_senior_citizen:
    print(name, "is a senior citizen.")
else:
    print(name, "is not a senior citizen.")
```

Description:

This program prompts the user to enter a person's name and year of birth. It uses the current year obtained from the `date` module to calculate the person's age. The `check_senior_citizen` function takes the name and year of birth as input and determines whether the person is a senior citizen based on their age (60 or above).

Output:

Sample 1:

```
Enter person's name: Sanyo
Enter year of birth: 1978
Sanyo is not a senior citizen.
```

Sample 2:

```
Enter person's name: Saroj
Enter year of birth: 1960
Saroj is a senior citizen.
```

9) What are functions? Explain Python function with parameters and return statements.

Solution:

A function is a block of reusable code that performs a specific task.

In Python, functions are defined using the `def` keyword, followed by the function name, parentheses for parameters (if any), a colon, and an indented block of code that constitutes the function body.

Here's an example of a Python function with parameters and a return statement:

```
def greet(name):
    greeting = "Hello, " + name + "!"
    return greeting

# Calling the function
result = greet("Alice")
print(result)
```

Output:

```
Hello, Alice!
```

In the above example, we define a function called `greet` that takes a single parameter `name`. To use the function, we call it with an argument, in this case, "Alice". The function executes its code with the provided argument and returns the result. The returned value is then assigned to the `result` variable. Finally, we print the value of `result`, which outputs "Hello, Alice!" to the console.

Functions with parameters allow us to create more flexible and reusable code. By accepting different arguments, the same function can be used to perform similar operations on various inputs.

The `return` statement is used to specify the value that should be sent back from the function to the caller. It terminates the function execution and provides the output. The returned value can be stored in a variable, used in an expression, or passed as an argument to another function.

10) What will be the output of the following Python code?

a)

```
def update_global_variable():

      global eggs

    eggs = 'spam'


# main program

eggs = 'bacon'

print("Before function call - eggs:", eggs)

update_global_variable()

print("After function call - eggs:", eggs)
```

OUTPUT:

```
Before function call - eggs: bacon

After function call - eggs: spam
```

b)

```
def function_bacon():
    eggs = 'bacon'
    print("Inside the function - eggs:", eggs)

# main program
eggs = 'spam'
print("Before function call - eggs:", eggs)
function_bacon()
print("After function call - eggs:", eggs)
```

OUTPUT:

```
Before function call - eggs: spam
Inside the function - eggs: bacon
After function call - eggs: spam
```

c)

```
def quiz_example_1():
     global variable_1
     variable_1 = 10
     variable_2 = 20

# main program
variable_1 = 5
variable_2 = 15
quiz_example_1()
print(variable_1)
print(variable_2)
```

OUTPUT:

b) 10 15

d)

```
def quiz_example_2():
    variable_1 = 5
    print("Inside function:", variable_1)

# main program
variable_1 = 10
quiz_example_2()
print("Outside function:", variable_1)
```

OUTPUT:

```
        Inside function: 5
        Outside function: 10
```

e)

```
def quiz_example_3():
    global variable_1
    variable_1 = 5

def another_function():
    variable_2 = 10
    print("Inside another function:", variable_2)

# main program
variable_1 = 10
variable_2 = 20
quiz_example_3()
another_function()
print("Outside functions:", variable_1)
print("Outside functions:", variable_2)
```

OUTPUT:

```
Inside another function: 10

Outside functions: 5

Outside functions: 20
```

f)

```
def update_list():
   global my_list
   my_list.append(4)

# main program
my_list = [1, 2, 3]
print("Before function call - my_list:", my_list)
update_list()
print("After function call - my_list:", my_list)
```

OUTPUT:

```
Before function call - my_list: [1, 2, 3]
After function call - my_list: [1, 2, 3, 4]
```

11) Write a Python function called "calculate_average" that takes a list of numbers as an input and returns the average (mean) of those numbers.

Solution:

```
# This is a function to calculate the average of a list of numbers.

def calculate_average(numbers):

   total = sum(numbers)        # Calculate the sum of the numbers in the list.

   count = len(numbers)        # Determine the count of numbers in the list.

   average = total / count     # Calculate the average by dividing the total by the count.

   return average              # Return the calculated average.


# Main program

numbers_list = [2, 4, 6, 8, 10]              # Example list of numbers

result = calculate_average(numbers_list)              # Call the calculate_average function

print("The average is:", result)               # Print the calculated average.
```

Description:

The program defines a function called "calculate_average" that calculates the average of a list of numbers. It uses the sum() function to calculate the total sum of the numbers and the len() function to determine the count of numbers. Then, it divides the total by the count to get the average.

OUTPUT:

The average is: 6.0

12) Explain string concatenation and string replication with one suitable example for each.

Solution:

String concatenation refers to the process of combining two or more strings into a single string. This can be done using the `+` operator, which joins the strings together.

Example of string concatenation:

```
first_name = "John"
last_name = "Doe"

full_name = first_name + " " + last_name
print(full_name)
```

Output:

```
John Doe
```

In this example, the variables `first_name` and `last_name` store the strings "John" and "Doe" respectively. By using the `+` operator, we can concatenate these strings with a space in between, resulting in the string "John Doe". The `print()` function is then used to display the concatenated string.

String replication, on the other hand, involves creating a new string by repeating an existing string a certain number of times. This can be achieved using the `*` operator.

Example of string replication:

```
message = "Hello!"

repeated_message = message * 3
print(repeated_message)
```

Output:

```
Hello!Hello!Hello!
```

In this example, the variable `message` contains the string "Hello!". By using the `*` operator and specifying the number 3, we replicate the string three times. The resulting string is "Hello!Hello!Hello!", which is then printed using the `print()` function.

13) What are Comparison and Boolean operators? List all the Comparison and Boolean operators in Python and explain the use of these operators with suitable examples.

Solution:

Comparison operators are used to compare two values or expressions and determine the relationship between them. These operators evaluate to either True or False, indicating whether the comparison is true or false.

The following are the comparison operators in Python:

1. `==` (Equal to): Checks if two values are equal.
2. `!=` (Not equal to): Checks if two values are not equal.
3. `>` (Greater than): Checks if the left operand is greater than the right operand.

4. `<` (Less than): Checks if the left operand is less than the right operand.
5. `>=` (Greater than or equal to): Checks if the left operand is greater than or equal to the right operand.
6. `<=` (Less than or equal to): Checks if the left operand is less than or equal to the right operand.

Example of comparison operators:

```
x = 5
y = 10

print(x == y)     # Output: False
print(x != y)     # Output: True
print(x > y)      # Output: False
print(x < y)      # Output: True
print(x >= y)    # Output: False
print(x <= y)    # Output: True
```

In this example, we compare the values of `x` and `y` using different comparison operators. Each comparison evaluates to either True or False, depending on the relationship between the values.

Boolean operators, also known as logical operators, are used to combine or manipulate Boolean values (True or False). These operators allow you to perform logical operations on the results of comparison operations.

The following are the Boolean operators in Python:

1. `and`: Returns True if both operands are True.
2. `or`: Returns True if either of the operands is True.
3. `not`: Returns the opposite Boolean value of the operand.

Example of Boolean operators:

```
x = 5
y = 10
z = 3

print(x < y and y < z)  # Output: False
print(x < y or y < z)  # Output: True
print(not x < y)  # Output: False
```

These operators are fundamental in controlling the flow of programs, making decisions, and performing logical operations in Python.

14) Explain importing of modules into an application in Python with syntax and a suitable programming example.

Solution:

Importing modules in Python programming allows us to use the functionalities present in external code libraries. Here's an explanation of how to import modules into your application with syntax and a suitable programming example:

Syntax for importing a module:

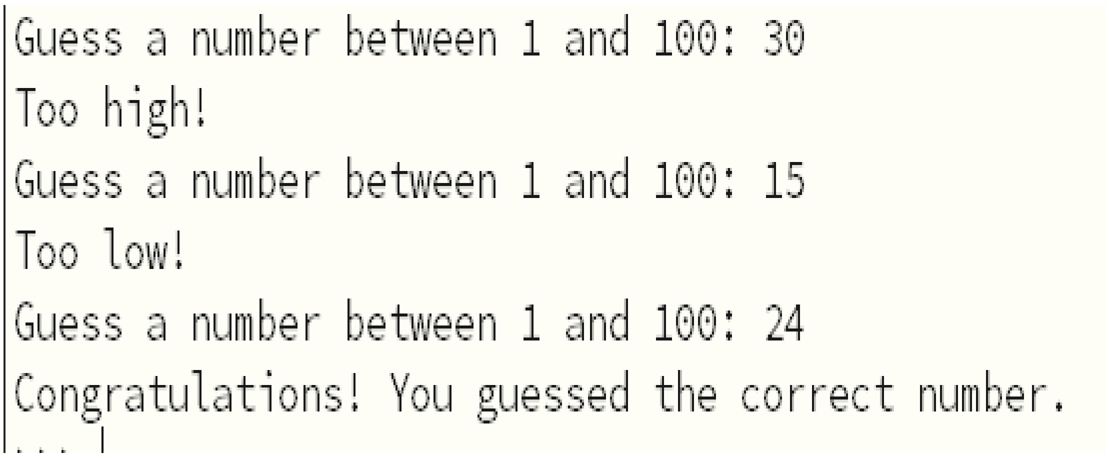
```
import module_name
```

Example:

Let us write a Python program that generates a random number between 1 and 100. The program should prompt the user to guess the number and provide feedback if the guess is too high or too low. The game should continue until the user correctly guesses the number.

```
import random
secret_number = random.randint(1,100)
guess = 0
while guess != secret_number:
    guess = int(input("Guess a number between 1 and 100: "))
    if guess > secret_number:
        print("Too high!")
    elif guess < secret_number:
        print("Too low!")
    else:
        print("Congratulations! You guessed the correct number.")
```

OUTPUT:



```
Guess a number between 1 and 100: 30
Too high!
Guess a number between 1 and 100: 15
Too low!
Guess a number between 1 and 100: 24
Congratulations! You guessed the correct number.
```

15) What is a function in Python? Write the syntax for defining a function.

Solution:

A function in Python is a reusable block of code that performs a specific task. It allows you to break down your code into smaller, modular pieces, making it easier to read, understand, and maintain.

The syntax for defining a function in Python is as follows:

```
def function_name(parameters):
  # Function body
  # Code to be executed
  # ...
  # Optional return statement
  return value
```

- The keyword def is used to indicate the start of a function definition.
- function_name is the name given to the function. It should follow the naming conventions for variables.
- parameters (optional) are inputs that the function can accept. Multiple parameters are separated by commas. If there are no parameters, empty parentheses are used.
- The colon (:) marks the end of the function header and the start of the function body.
- The function body is indented, typically with four spaces, and contains the code that will be executed when the function is called.
- Within the function body, you write the logic and operations that the function performs.
- Optionally, a return statement can be used to specify the value or values that the function will output. If there is no return statement, the function will return None by default.

For example, let's define a simple function called greet that takes a name as a parameter and prints a greeting message:

```
def greet(name):
    print("Hello, " + name + "! How are you?")


# Calling the function
greet("Alice")
```

16) What are try and except statements used for in exception handling in Python?

Solution:

In exception handling, the try and except statements are used to handle errors in a program. The code that is likely to generate an error is enclosed within the try clause. If an error occurs during the execution of the code inside the try clause, the program moves to the start of the statements in the except clause to handle the error. This helps in detecting errors, handling them, and allowing the program to continue running without terminating abruptly.

In the given example, the function `spam()` is defined to perform division. However, a ZeroDivisionError can occur if the second argument is 0. To handle this potential error, the try-except block is used. Inside the try block, the `spam()` function is called multiple times with different arguments. If a ZeroDivisionError occurs during the execution of `spam()`, the program jumps to the except block, which prints 'Error: Invalid argument.' Instead of terminating the program abruptly, it continues to execute the subsequent statements.

Example:

```
def spam(num1, num2):
    return num1/num2


try:
    print(spam(12, 2))     # Normal division, no error
    print(spam(48, 12))    # Normal division, no error
    print(spam(12, 0))     # ZeroDivisionError occurs
    print(spam(5, 1))      # Normal division, no error
```

```
except ZeroDivisionError:
    print('Error: Invalid argument.')
```

In the given example, the third call to `spam(12, 0)` results in a ZeroDivisionError, which is caught by the except block. Instead of terminating the program, it executes the except block and prints 'Error: Invalid argument.' This demonstrates the handling of errors using try and except statements in Python's exception handling mechanism.

OUTPUT:

```
6.0
4.0
Error: Invalid argument.
5.0
```

17) Write a program that takes a number as input and determines whether it is positive, negative, or zero.

Solution:

```
# Program to determine whether a number is positive, negative, or zero
number = float(input("Enter a number: "))       # Input a number from the user
# Check if the number is greater than zero and print the appropriate message
if number > 0:
     print("The number is positive.")
elif number < 0:          # Check if the number is less than zero and print the appropriate message
     print("The number is negative.")
else:          # If neither condition is true, then print the message that number is zero
    print("The number is zero.")
```

Description:

This program takes a number as input from the user and determines whether it is positive, negative, or zero. It uses an `if-elif-else` conditional statement to check the value of the input number. If the number is greater than zero, it prints a message stating that the number is positive. If the number is less than zero, it prints a message stating that the number is negative. If neither of these conditions is true, it means the number is zero, and it prints a message indicating that the number is zero.

Sample Output 1:

```
Enter a number: 5
The number is positive.
```

Sample Output 2:

Enter a number: -3.5

The number is negative.

18) Name any three different data types in Python and explain each.

Solution:

In Python, there are several built-in data types. Here are three commonly used data types:

1) Integer (int): Integers are used for numeric computations and represent whole numbers.

The integer data type represents whole numbers without any decimal points.

Examples: -5, 0, 10, 100

Integers can be positive or negative and have no limit on their size.

2) String (str): Strings are used to store and manipulate textual data, such as words or sentences.

The string data type represents a sequence of characters enclosed in single quotes ('') or double quotes ("").

Examples: "Hello", 'Python', "12345"

Strings are used to store textual data and can contain letters, numbers, symbols, and whitespace.

3) Boolean (bool): Booleans are used for logical operations and help determine the truth value of an expression.

The boolean data type represents a logical value that can be either True or False.

Examples: True, False

Booleans are often used in conditional statements and logical operations to control the flow of a program.

Python is a dynamically typed language, hence we don't need to explicitly declare the data type of a variable. Python determines the data type based on the value assigned to the variable.

19) Critique the importance of indentation in Python and how it affects program execution.

Solution:

In Python, indentation is used to indicate the grouping and hierarchy of statements within control structures such as loops, conditionals, and function definitions. Indentation in Python is important because:

Readability and Code Organization:

- Indentation improves the readability of the code by visually representing the logical structure of the program.
- Proper indentation makes the code more organized and easier to maintain, debug, and collaborate on.

Block Delimiters:

- In many programming languages, curly braces or keywords are used to delimit code blocks. However, in Python, indentation is used as a block delimiter.
- Blocks of code with the same indentation level are considered part of the same block, while a decrease in indentation marks the end of a block.

Program Execution:

- Indentation directly affects the execution of the program. Incorrect indentation can lead to syntax errors or alter the logic of the code.

Code Flow and Control Structures:

- Indentation is essential for control structures such as if-else statements, for loops, while loops, and function definitions.
- Proper indentation is necessary to accurately define the body of these control structures.

20) How does a function differ from a variable in Python? Name some built-in functions in Python.

Solution:

Difference between a function and a variable:

- A variable is used to store and manipulate data. It holds a value that can be accessed, modified, or reassigned throughout the program.
- A function, on the other hand, is a block of code that performs a specific task or set of operations. It can take inputs (parameters), perform actions, and optionally return a result.

Hence we can say that variable stores data, while a function performs actions or computations.

Built-in functions in Python: Python comes with a rich set of built-in functions that are readily available for use. Some commonly used built-in functions in Python include:

1. print(): Used to display output on the console.
2. input(): Takes user input from the console.
3. len(): Returns the length of an object (e.g., string, list).
4. type(): Returns the type of an object.
5. range(): Generates a sequence of numbers.
6. sum(): Calculates the sum of elements in an iterable.
7. max(): Returns the maximum value from a sequence.
8. min(): Returns the minimum value from a sequence.
9. abs(): Returns the absolute value of a number.
10. round(): Rounds a floating-point number to a specified decimal place.

21) Define a variable in Python and give an example.

Solution:

In Python, a variable is a named container that can hold a value. It allows you to store and manipulate data within your program. To define a variable in Python, you simply choose a name for the variable and assign a value to it using the assignment operator (=).

Here's an example of defining a variable in Python:

```
# Define a variable named 'message' and assign a string value to it
message = "Hello, world!"

# Print the value of the variable
print(message)
```

In the above example, a variable named 'message' is defined and assigned the value "Hello, world!". The equal sign (=) is used to assign the value to the variable. Then, the value of the 'message' variable is printed using the 'print()' function, which will output "Hello, world!" to the console.

Variable names must follow certain naming rules such as starting with a letter or underscore and consisting of letters, digits, or underscores. Variables can hold various types of data, including numbers, strings, lists, dictionaries, and more.

22) Explain the difference between a string and a number in Python.

Solution:

In Python, a string is a sequence of characters enclosed in either single quotes ('') or double quotes (""). Strings are used to represent textual data and are treated as a series of individual characters. They can contain letters, numbers, symbols, and whitespace.

Here's an example of a string in Python:

For example,

```
my_string = "Hello, World!"
```

In the above example, my_string is a variable that holds a string value "Hello, World!".

On the other hand, a number in Python represents numerical values and can be of different types, such as integers (whole numbers), floating-point numbers (decimal numbers), or complex numbers.

Here are a few examples of numbers in Python:

```
my_integer = 42

my_float = 3.14

my_complex = 2 + 3j
```

The main difference between strings and numbers in Python is their representation and the operations that can be performed on them. Strings are treated as text and support various string-specific operations like concatenation, slicing, and string methods. Numbers, on the other hand, are used for mathematical operations and support arithmetic operations such as addition, subtraction, multiplication, and division.

Here's an example that showcases the difference between a string and a number in Python:

```
my_string = "42"

my_number = 42

print(my_string + my_string)  # Concatenates two strings: "4242"

print(my_number + my_number)  # Adds two numbers: 84
```

In the above example, when the + operator is used with strings, it concatenates the two strings together, whereas when it is used with numbers, it performs addition.

23) Explain the concept of an "expression" in Python with example.

Solution:

In Python, an expression is a combination of values, variables, operators, and function calls that evaluates to a single value. Expressions can be as simple as a single value or as complex as a combination of multiple operations.

Here's an example of an expression in Python:

```
x = 5

y = 3

result = x + y * 2

print(result)
```

The expression result = x + y * 2 assigns the value of the expression x + y * 2 (which is 11) to the variable result.

The expression print(result) calls the print() function with the value of the variable result as an argument. It outputs the value 11 to the console.

Expressions can also involve other operators, function calls, and more complex combinations of values and variables. They are the building blocks of any program and are used to perform calculations, manipulate data, and make decisions based on the evaluated results.

24) Identify and explain the components of a Python statement.

Solution:

In Python, a statement is a line of code that performs a specific action or operation. A statement typically consists of one or more components as follows:

1) Keywords: Keywords are reserved words in Python that have predefined meanings and cannot be used as variable names. Examples of keywords include if, for, while, def, class, import, etc. Keywords define the structure and flow of the program.

2) Variables: Variables are used to store and manipulate data. They are named containers that hold values of different types, such as integers, strings, lists, etc. Variables are assigned values using the assignment operator (=) and can be used in expressions and statements.

3) Operators: Operators are symbols or special characters that perform specific operations on one or more values. Python supports various types of operators, such as arithmetic operators (+, -, *, /), comparison operators (>, <, ==, !=), assignment operators (=, +=, -=), logical operators (and, or, not), etc. Operators help in performing calculations, comparisons, and logical operations.

4) Functions: Functions are reusable blocks of code that perform specific tasks. They take input arguments (if any), perform operations, and may return a value. Python provides built-in functions like print(), len(), input(), etc. Additionally, you can define your own functions to modularize and organize your code.

5) Control Flow Structures: Control flow structures determine the order in which statements are executed. Common control flow structures include if-else statements, loops (for and while), and function calls. They allow you to make decisions, repeat code blocks, and control the flow of your program based on certain conditions.

These components come together to form complete Python statements that instruct the interpreter on what actions to perform.