1.

a) Describe Arithmetic Operators, Assignment Operators, Comparison Operators, Logical Operators and Bitwise Operators in detail with examples.

Ans:

Arithmetic operator:

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

In [1]:
```python
# for example:
a= 20
b= 4
print('Addition: ',a+b)
print('Substraction: ',a-b)
print('Multiplication: ',a*b)
print('Division: ',a/b)
print('Modulus: ',a%b)
print('Exponentiation: ',a**b)
print('Floor Division: ',a//b)
```

```
Addition:  24
Substraction:  16
Multiplication:  80
Division:  5.0
Modulus:  0
Exponentiation:  160000
Floor Division:  5
```

Assignment operator:

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operator:

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators:

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Bitwise operators:

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

b) Explain the Identifiers, Keywords, Statements, Expressions, and Variables in Python programming language with examples.

- Identifiers:
  - Identifier is a user-defined name given to a variable, function, class, module, etc. The identifier is a combination of character digits and an underscore. They are case-sensitive i.e., 'num' and 'Num' and 'NUM' are three different identifiers in python. It is a good programming practice to give meaningful names to identifiers to make the code understandable.
  - It cannot be a reserved python keyword.
  - It should not contain white space.
  - It can be a combination of A-Z, a-z, 0-9, or underscore.
  - It should start with an alphabet character or an underscore (_).
  - It should not contain any special character other than an underscore (_).
- Example:
  - Valid identifiers:

var1

_var1

_1_var

var_1

- Keywords:
  - Python Keywords are some predefined and reserved words in python that have special meanings.
  - Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, or variable name.
  - All the keywords in python are written in lowercase except True and False.
  - There are 35 keywords in Python 3.11.

In [1]:
```python
# keywords are:
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

- Statements:
  - A Python statement is an instruction that the Python interpreter can execute.
  - There are different types of statements in Python language as Assignment statements, Conditional statements, Looping statements, etc.
  - For example:

In [2]:
```python
# assignment statement:
count = 10
```

- Expressions:
  - A combination of operands and operators is called an expression. The expression in Python produces some value or result after being interpreted by the Python interpreter. An expression in Python is a combination of operators and operands.

In [3]:
```python
# example:

x = 25          # a statement
x = x + 10      # an expression
print(x)
```

```
35
```

- Variables:
  - Python Variable is containers that store values.
  - A Python variable name must start with a letter or the underscore character.
  - A Python variable name cannot start with a number.
  - A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
  - Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
  - The reserved words(keywords) in Python cannot be used to name the variable in Python.

In [4]:
```python
# for example:
Var = "Student"
print(Var)
```

```
Student
```

C) Explain the basic data types available in Python with examples.

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

In [5]:
```python
# for example
x = 5
print(type(x))
```

```
<class 'int'>
```

2)

a) Write Python Program to reverse a number and also find the Sum of digits in the reversed number. Prompt the user for input.

In [9]:
```python
# Prompt the user to enter a number
num = int(input("Enter a number: "))
# Initialize variables
reverse_num = 0
sum_of_digits = 0
# Reverse the number and find the sum of digits in the reversed number
while num > 0:
    digit = num % 10
    reverse_num = reverse_num * 10 + digit
    sum_of_digits += digit
    num //= 10
# Display the reversed number and the sum of digits in the reversed number
print("Reversed number:", reverse_num)
print("Sum of digits in the reversed number:", sum_of_digits)
```

```
Enter a number: 123
Reversed number: 321
Sum of digits in the reversed number: 6
```

b) Write Pythonic code to check if a given year is a leap year or not.

In [13]:
```python
def is_leap_year(year):
# Returns True if the given year is a leap year, False otherwise.
   if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
    return True
   else:
    return False
is_leap_year(2000)
```

Out[13]:
```
True
```

c) Write Python program to find the GCD of two positive numbers.

In [14]:
```python
# Define a function to find the GCD of two numbers
def gcd(a, b):
# Returns the GCD (Greatest Common Divisor) of two positive integers a and b.
  if b == 0:
    return a
  else:
    return gcd(b, a % b)
# Prompt the user to enter two positive integers
a = int(input("Enter the first positive integer: "))
b = int(input("Enter the second positive integer: "))
# Call the gcd function to find the GCD of a and b
result = gcd(a, b)
# Display the result
print("The GCD of", a, "and", b, "is", result)
```

```
Enter the first positive integer: 12
Enter the second positive integer: 24
The GCD of 12 and 24 is 12
```

3)

a) Write Python code to determine whether the given string is a Palindrome or not using slicing.

In [17]:
```python
# Prompt the user to enter a string
string = input("Enter a string: ")
# convert string to lowercase
string=str.casefold(string)
# Reverse the string using slicing
reverse_string = string[::-1]
# Check if the string is equal to its reverse
if string == reverse_string:
     print("The string is a palindrome.")
else:
     print("The string is not a palindrome.")
```

```
Enter a string: Malayalam
The string is a palindrome.
```

b) Explain the use of join() and split() string methods with examples. Describe why strings are immutable with an example.

In [19]:
```python
delimiter = " "
string_sequence = ["Hello", "World"]
joined_string = delimiter.join(string_sequence)
print(joined_string)
string = "Hello World"
delimiter = " "
split_string = string.split(delimiter)
print(split_string)
```

```
Hello World
['Hello', 'World']
```

Why strings are immutable in python?

Ans:

```
An immutable object is one that, once created, will not change in its lifetime.
```

In [21]:
```python
# for example
name= 'Rishabh'
name[0]='H'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-8d0ecf69e61c> in <module>
      1 # for example
      2 name= 'Rishabh'
----> 3 name[0]='H'

TypeError: 'str' object does not support item assignment
```

c) Write Python program to count the total number of vowels, consonants and blanks in a String.

In [23]:
```python
# Prompt the user to enter a string
string = input("Enter a string: ")
# Initialize variables for counting vowels, consonants, and blanks
vowels = 0
consonants = 0
blanks = 0
# Convert the string to lowercase
string = string.lower()
# Loop through each character in the string
for char in string:
# Check if the character is a vowel
   if char in "aeiou":
    vowels += 1
# Check if the character is a consonant
   elif char.isalpha():
    consonants += 1
# Check if the character is a blank space
   elif char == " ":
    blanks += 1
# Display the results
print("Number of vowels:", vowels)
print("Number of consonants:", consonants)
print("Number of blanks:", blanks)
```

```
Enter a string: Favourite
Number of vowels: 5
Number of consonants: 4
Number of blanks: 0
```

4)

a) Write Python program to add two matrices and also find the transpose of the resultant matrix.

In [25]:
```python
# Initialize the matrices
matrix1 = [[1, 4, 3], [4, 3, 6], [4, 8, 9]]
matrix2 = [[9, 6, 7], [1, 5, 4], [3, 8, 1]]
# Create a matrix to store the result
result = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
# Add the matrices and store the result in the result matrix
for i in range(len(matrix1)):
    for j in range(len(matrix1[0])):
        result[i][j] = matrix1[i][j] + matrix2[i][j]
# Print the result matrix
print("Resultant matrix:")
for row in result:
    print(row)
# Find the transpose of the resultant matrix
transpose = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
for i in range(len(result)):
    for j in range(len(result[0])):
        transpose[j][i] = result[i][j]
# Print the transpose of the resultant matrix
print("Transpose of the resultant matrix:")
for row in transpose:
    print(row)
```

```
Resultant matrix:
[10, 10, 10]
[5, 8, 10]
[7, 16, 10]
Transpose of the resultant matrix:
[10, 5, 7]
[10, 8, 16]
[10, 10, 10]
```

b) Input five integers (+ve and −ve). Write Pythonic code to find the sum of negative numbers, positive numbers and print them. Also, find the average of all the numbers and numbers above average.

In [26]:
```python
# Prompt the user to input five integers
print("Enter five integers:")
num_list = []
for i in range(5):
    num_list.append(int(input()))
# Calculate the sum of negative and positive numbers
negative_sum = 0
positive_sum = 0
for num in num_list:
    if num < 0:
        negative_sum += num
    elif num > 0:
        positive_sum += num
# Calculate the average of all the numbers
average = sum(num_list) / len(num_list)
# Find the numbers above the average and calculate their average
above_average_nums = [num for num in num_list if num >average]
above_average_average = sum(above_average_nums) /len(above_average_nums) if above_average_nums else 0
# Print the results
print("Sum of negative numbers:", negative_sum)
print("Sum of positive numbers:", positive_sum)
print("Average of all numbers:", average)
print("Average of numbers above average:",above_average_average)
```

```
Enter five integers:
-15
20
5
-8
5
Sum of negative numbers: -23
Sum of positive numbers: 30
Average of all numbers: 1.4
Average of numbers above average: 10.0
```

c) Write Pythonic code to find Mean, Variance and Standard Deviation for a list of numbers.

In [27]:
```python
import math
# Prompt the user to input a list of numbers
print("Enter a list of numbers separated by spaces:")
num_list = list(map(float, input().split()))
# Calculate the mean
mean = sum(num_list) / len(num_list)
# Calculate the variance
variance = sum([(x - mean) ** 2 for x in num_list]) /len(num_list)
# Calculate the standard deviation
std_dev = math.sqrt(variance)
# Print the results
print("Mean:", mean)
print("Variance:", variance)
print("Standard Deviation:", std_dev)
```

```
Enter a list of numbers separated by spaces:
12 25 47 35 48
Mean: 33.4
Variance: 185.84
Standard Deviation: 13.632314550361578
```

5)

a) Discuss the relation between tuples and lists, tuples and dictionaries in detail.

- Tuples vs Lists

The main difference between tuples and lists is that tuples are immutable, while lists are mutable. This means that once a tuple is created, its contents cannot be changed, while a list can be modified by adding, removing, or changing elements.

- Tuples vs Dictionaries

Tuples and dictionaries are also different data structures that serve different purposes. Tuples are ordered collections of values, while dictionaries are unordered collections of key-value pairs. Here are some key differences between tuples and dictionaries:

b) Write Python program to swap two numbers without using Intermediate/Temporary variables. Prompt the user for input.

In [29]:
```python
# Prompt the user for input
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
# Print the original values
print("Before swapping: num1 =", num1, "and num2 =",
num2)
# Swap the values without using intermediate variables
num1, num2 = num2, num1
# Print the swapped values
print("After swapping: num1 =", num1, "and num2 =", num2)
```

```
Enter the first number: 45
Enter the second number: 68
Before swapping: num1 = 45 and num2 = 68
After swapping: num1 = 68 and num2 = 45
```

c) Write a program that accepts a sentence and calculate the number of digits, uppercase and lowercase letters.

In [31]:
```python
# Prompt the user for input
sentence = input("Enter a sentence: ")
# Initialize counters
num_digits = 0
num_uppercase = 0
num_lowercase = 0
# Loop through each character in the sentence
for char in sentence:
    if char.isdigit():
        num_digits += 1
    elif char.isupper():
        num_uppercase += 1
    elif char.islower():
        num_lowercase += 1
# Print the results
print("Number of digits:", num_digits)
print("Number of uppercase letters:", num_uppercase)
print("Number of lowercase letters:", num_lowercase)
```

```
Enter a sentence: Rishabh276
Number of digits: 3
Number of uppercase letters: 1
Number of lowercase letters: 6
```

6)

a) Write Pythonic code to sort a sequence of names according to their alphabetical order without using sort() function.

In [33]:
```python
# Prompt the user for input
names = input("Enter a sequence of names separated by spaces: ")
# Convert the string of names into a list
names_list = names.split()
# Bubble sort algorithm to sort the names
for i in range(len(names_list)):
    for j in range(len(names_list) - 1):
        if names_list[j] > names_list[j+1]:
        # Swap the elements
           names_list[j], names_list[j+1] = names_list[j+1],names_list[j]
# Print the sorted list of names
print("Sorted names:", end=" ")
for name in names_list:
    print(name, end=" ")
```

```
Enter a sequence of names separated by spaces: Rishabh Sayan Akash Raj Indrani
Sorted names: Akash Indrani Raj Rishabh Sayan
```

b) Discuss zip() function with an example.

- The zip() function in Python is used to combine two or more iterables (e.g. lists, tuples, sets, etc.) into a single iterable object, where each element of the new iterable contains the corresponding elements from each of the input iterables.

- The resulting iterable is a zip object, which can be converted to other iterables (e.g. list, tuple) or used in loops.

In [36]:
```python
# Define two lists
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
# Use zip() to combine the two lists
name_age = zip(names, ages)
# Print the resulting zip object
print(name_age)
# Convert the zip object to a list and print it
print(list(name_age))
# Iterate over the zip object using a for loop and print each element
for na in zip(names, ages):
    print(na)
```

```
<zip object at 0x0000018A6D508300>
[('Alice', 25), ('Bob', 30), ('Charlie', 35)]
('Alice', 25)
('Bob', 30)
('Charlie', 35)
```

c) Illustrate the following Set methods with an example.

a) intersection() b) union() c) issubset() d) difference() e) update() f) discard()

In [37]:
```python
# a)
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)
```

```
{'apple'}
```

In [38]:
```python
# b)
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.union(y)

print(z)
```

```
{'apple', 'google', 'banana', 'cherry', 'microsoft'}
```

In [39]:
```python
# c)
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z)
```

```
True
```

In [40]:
```python
# d)
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.difference(y)

print(z)
```

```
{'cherry', 'banana'}
```

In [41]:
```python
# e)
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.update(y)

print(x)
```

```
{'apple', 'google', 'banana', 'cherry', 'microsoft'}
```

In [42]:
```python
# f)
fruits = {"apple", "banana", "cherry"}

fruits.discard("banana")

print(fruits)
```

```
{'apple', 'cherry'}
```

9)

a) Write Python Program to Calculate Area and Perimeter of different Shapes using Polymorphism.

In [43]:
```python
import math
class Shape:
    def area(self):
        pass
    def perimeter(self):
        pass
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
    def perimeter(self):
        return 2 * (self.length + self.width)
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return math.pi * (self.radius ** 2)
    def perimeter(self):
        return 2 * math.pi * self.radius
class Square(Rectangle):
    def __init__(self, side):
        super().__init__(side, side)
# Example usage
rectangle = Rectangle(4, 5)
print("Rectangle Area:", rectangle.area())
print("Rectangle Perimeter:", rectangle.perimeter())
circle = Circle(3)
print("Circle Area:", circle.area())
print("Circle Perimeter:", circle.perimeter())
square = Square(4)
print("Square Area:", square.area())
print("Square Perimeter:", square.perimeter())
```

```
Rectangle Area: 20
Rectangle Perimeter: 18
Circle Area: 28.274333882308138
Circle Perimeter: 18.84955592153876
Square Area: 16
Square Perimeter: 16
```

b) Illustrate the diamond problem in Python programming language with an example.

In [44]:
```python
# define the size (no. of columns)
# must be odd to draw proper diamond shape
size = 11

# initialize the spaces
spaces = size

# loops for iterations to create worksheet
for i in range(size//2+2):
    for j in range(size):

        # condition to left space
        # condition to right space
        # condition for making diamond
        # else print *
        if j < i-1:
            print(' ', end=" ")
        elif j > spaces:
            print(' ', end=" ")
        elif (i == 0 and j == 0) | (i == 0 and j == size-1):
            print(' ', end=" ")
        else:
            print('*', end=" ")

    # increase space area by decreasing spaces
    spaces -= 1

    # for line change
    print()
```

```
  * * * * * * * * *
* * * * * * * * * * *
  * * * * * * * * *
    * * * * * * *
      * * * * *
        * * *
          *
```

10)

a) Write Python Program to Demonstrate the Construction of Method Resolution Order in Python.

In [45]:
```python
class A:
    def method(self):
        print("A method called")
class B(A):
    pass
class C(A):
    def method(self):
        print("C method called")
class D(B, C):
    pass
# Print the MRO of class D
print(D.mro())
```

```
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
```

b) Consider a Rectangle Class and Create Two Rectangle Objects. Write Python program to Check Whether the Area of the First Rectangle is Greater than Second by Overloading > Operator.

In [46]:
```python
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
    def __gt__(self, other):
        return self.area() > other.area()
# Create two Rectangle objects
rect1 = Rectangle(4, 5)
rect2 = Rectangle(3, 6)
# Compare their areas using the > operator
if rect1 > rect2:
    print("The area of the first rectangle is greater than the second.")
else:
    print("The area of the second rectangle is greater than the first.")
```

```
The area of the first rectangle is greater than the second.
```

END