1.Explain different operators used.
Certainly! supports a variety of operators, which are symbols used to perform operations on variables and values. Here are some of the commonly used operators in :

1. Arithmetic Operators:
`+` (Addition): Adds two operands.
result_add = 5 + 3 result_add = 8
`` (Subtraction): Subtracts the right operand from the left operand.
result_sub = 7 2 result_sub = 5
`*` (Multiplication): Multiplies two operands.
result_mul = 4 * 6 result_mul = 24
`/` (Division): Divides the left operand by the right operand (returns a float).
result_div = 9 / 3 result_div = 3.0
`%` (Modulus): Returns the remainder of the division.
result_mod = 11 % 3 result_mod = 2

2. Comparison Operators:
`==` (Equal to): True if the operands are equal.
`!=` (Not equal to): True if the operands are not equal.
`>` (Greater than): True if the left operand is greater than the right operand.
`<` (Less than): True if the left operand is less than the right operand.
`>=` (Greater than or equal to): True if the left operand is greater than or equal to the right operand.
`<=` (Less than or equal to): True if the left operand is less than or equal to the right operand.

3. Logical Operators:
`and` (Logical AND): True if both operands are true.
`or` (Logical OR): True if at least one operand is true.
`not` (Logical NOT): True if the operand is false.

4. Assignment Operators:
`=` (Assign): Assigns the value on the right to the variable on the left.
`+=` (Add and Assign): Adds the right operand to the left operand and assigns the result to the left operand.
`=` (Subtract and Assign): Subtracts the right operand from the left operand and assigns the result to the left operand.
`*=` (Multiply and Assign): Multiplies the left operand by the right operand and assigns the result to the left operand.
`/=` (Divide and Assign): Divides the left operand by the right operand and assigns the result to the left operand.

5. Membership Operators:
`in`: True if a value is found in the sequence.
`not in`: True if a value is not found in the sequence.

6. Identity Operators:
`is`: True if both variables refer to the same object.
`is not`: True if both variables do not refer to the same object.

2. What are the decision control statements in ?

In , decision control statements allow you to control the flow of your program based on certain conditions.

The primary decision control statements are:

1. if Statement:

The `if` statement is used to execute a block of code if a given condition is true.

Syntax:

```
if condition:
  Code to execute if the condition is true
```

Example:

```
x = 10
if x > 5:
   print("x is greater than 5")
```

2. if-else Statement:

The `if-else` statement is used to execute one block of code if the condition is true and another block if the condition is false.

Syntax:

```
if condition:
  Code to execute if the condition is true
else:
    Code to execute if the condition is false
```

Example:

```
x = 3
if x % 2 == 0:
   print("x is even")
else:
  print("x is odd")
```

3. if-elif-else Statement:

The `if-elif-else` statement is used when you have multiple conditions to check. It allows you to provide alternative blocks of code for each condition.

Syntax:

```
if condition1:
    Code to execute if condition1 is true
elif condition2:
    Code to execute if condition2 is true
else:
    Code to execute if none of the conditions are true
```

Example:

```
score = 75
if score >= 90:
   print("A")
elif score >= 80:
   print("B")
elif score >= 70:
   print("C")
else:
```

print("F")

4. Nested if Statements:

You can also have nested `if` statements, where one `if` statement is inside another. This is useful when you need to check multiple conditions in a specific order.

Example:

```
x = 10
if x > 5:
    print("x is greater than 5")
    if x == 10:
        print("x is equal to 10")
    else:
        print("x is not equal to 10")
    else:
        print("x is not greater than 5")
```

3. What are the different loops used in ? Explain break and continue.

In , there are two main types of loops: `for` loops and `while` loops. Additionally, the `break` and `continue` statements are used to control the flow within loops.

1. `for` Loop:

The `for` loop is used to iterate over a sequence (such as a list, tuple, string, or range) and execute a block of code for each item in the sequence.

Syntax:

```
for variable in sequence:
    Code to execute for each item in the sequence
```

Example:

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

2. `while` Loop:

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

Syntax:

```
while condition:
    Code to execute as long as the condition is true
```

Example:

```
count = 0
while count < 5:
print(count)
count += 1
```

Certainly! The `break` and `continue` statements are control flow statements used in loops in .

3. `break` Statement:

The `break` statement is used to terminate the loop prematurely, even if the loop condition is still true. It is often used when a certain condition is met, and you want to exit the loop immediately.

Syntax:

```
for variable in sequence:
    Code to execute
```

if condition: break

Example:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for number in numbers:
    if number == 5:
        print("Breaking the loop at 5.")
        break
    print(number)
```

2. `continue` Statement:

The `continue` statement is used to skip the rest of the code inside the loop for the current iteration and move on to the next iteration of the loop.

Syntax:

```
for variable in sequence:
    Code to execute
  if condition:
    continue
```

Example:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for number in numbers:
    if number % 2 == 0:
   print(f"Skipping even number: {number}")
   continue
    print(number)
```

Example with Both `break` and `continue`:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for number in numbers:
    if number % 2 == 0:
        print(f"Skipping even number: {number}")
        continue
    elif number == 7:
        print("Found 7, breaking loop.")
        break
    print(number)
```

4. **Explain string concatenation and replication.**

string concatenation involves combining two or more strings, while string replication involves creating a new string by repeating an existing string multiple times. Here are examples of both string concatenation and replication:

1. String Concatenation:

String concatenation is the process of combining two or more strings into a single string.

Syntax:

```
new_string = string1 + string2
```

Example:

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

2. String Replication: String replication involves creating a new string by repeating an existing string multiple times.

Syntax:

```
new_string = original_string * n
```

Example:

```
greeting = "Hello, "
repeated_greeting = greeting * 3
print(repeated_greeting)
```

5. **Explain exception handing .**

Exception handling in involves dealing with errors or exceptional situations that may occur during the execution of a program. provides a `try`, `except` block for handling exceptions.

Syntax:

```
try:   Code that might raise an exception
except ExceptionType as e:  Code to handle the exception
else:     Code to execute if no exception is raised
finally:    Code that will be executed no matter what (optional)
```

- The `try` block contains the code that might raise an exception.
- The `except` block catches and handles the specified exception type
- The `else` block is executed if no exception is raised in the `try` block.
- The `finally` block contains code that will be executed no matter what, whether an exception is raised or not (optional).

Example:

Let's consider an example where we try to divide two numbers

```
def divide_numbers(x, y):
   try:
      result = x / y
   except ZeroDivisionError as zd_error:     print(f"Error: {zd_error} - Cannot divide by zero.")
   except TypeError as type_error:   print(f"Error: {type_error} - Unsupported operation.")
   else: print(f"Result: {result}")
   finally:  print("This block is always executed.")
```

Example usage

```
divide_numbers(10, 2)   No exception
divide_numbers(10, 0)   Raises a ZeroDivisionError
divide_numbers("10", 2)   Raises a TypeError
```

6. Explain Augmented assignment operator.

Augmented assignment operators in are a shorthand way of performing an operation and assigning the result to a variable in a single step.

➔ **They combine an arithmetic or bitwise operation with assignment.**

➔ **Augmented assignment is concise and can make code more readable.**

<u>Here's a list of common augmented assignment operators:</u>

- **`+=`: Add and assign**
- **`-=`: Subtract and assign**
- **`*=`: Multiply and assign**
- **`/=`: Divide and assign**
- **`//=`: Floor divide and assign**
- **`%=`: Modulus and assign**
- **`**=`: Exponentiate and assign**
- **`&=`: Bitwise AND and assign**
- **`|=`: Bitwise OR and assign**
- **`^=`: Bitwise XOR and assign**
- **`<<=`: Left shift and assign**
- **`>>=`: Right shift and assign**

Syntax:

variable op= expression

Where `op` is one of the operators listed above.

Example:

x = 5

Equivalent to x = x + 3

x += 3 x is now 8

Equivalent to x = x * 2

x *= 2 x is now 16

Equivalent to x = x / 4

x /= 4 x is now 4.0

Equivalent to x = x % 3

x %= 3 x is now 1.0

7. Explain the list methods : append(),insert(),remove(),sort()

1. `append()`

The `append()` method is used to add an element to the end of the list.

Syntax:

```
list_name.append(element)
```

Example:

```
fruits = ['apple', 'banana', 'cherry']
fruits.append('orange')
print(fruits)
```

2. `insert()`

The `insert()` method is used to insert an element at a specified position in the list.

Syntax:

list_name.insert(index, element)

Example:

```
fruits = ['apple', 'banana', 'cherry']
fruits.insert(1, 'orange')
print(fruits)
```

3. `remove()`

The `remove()` method is used to remove the first occurrence of a specified element from the list.

Syntax:

```
list_name.remove(element)
```

Example:

```
fruits = ['apple', 'banana', 'cherry']
fruits.remove('banana')
print(fruits)
```

4. `sort()`

The `sort()` method is used to sort the elements of a list in ascending order. Optionally, you can specify the `reverse` parameter to sort in descending order.

Syntax:

```
list_name.sort(reverse=False)
```

Example:

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort()
print(numbers)
```

8. Differentiate between dictionary and list.

List	Dictionary
* Collection of various elements just like array in C++	* Collec^2 of elements in tagged structure as key-value pairs
* list is created placing all elements inside square brackets [], seperated by comas	* placing all key values inside curly brackets ({}) seperated by comma. ~~also each key & pair is seperated by semi-colon~~
* indices are integer value starting from 0	* The keys can be of any data type
* The order of the element is maintained	* No guarantee of maintaining the order
* lists are mutable,	* mutable but
* allow duplicate values	* does not allow
* Reverse method reverse list elements	* can not be reversed
* Slicing can be done	* slicing not done
* ex:- List = [10, 20, 30]	* Dict = {"ram": 26, "mary": 24}

9. Explain any 7 string methods.

1. `upper()`

The `upper()` method is used to convert all characters in a string to uppercase.

Syntax:

```
string.upper()
```

Example:

```
text = "hello world"
uppercase_text = text.upper()
print(uppercase_text)
```

2. `lower()`

The `lower()` method is used to convert all characters in a string to lowercase.

Syntax:

```
string.lower()
```

Example:

```
text = "Hello World"
lowercase_text = text.lower()
print(lowercase_text)
```

3. `strip()`

The `strip()` method is used to remove leading and trailing whitespaces from a string.

Syntax:

```
string.strip()
```

Example:

```
text = "   Python Programming   "
stripped_text = text.strip()
print(stripped_text)
```

4. `replace()`

The `replace()` method is used to replace a specified substring with another substring.

Syntax:

```
string.replace(old_substring, new_substring)
```

Example:

```
text = "Hello, World!"
new_text = text.replace("Hello", "Hi")
print(new_text)
```

5. `split()`

The `split()` method is used to split a string into a list of substrings based on a specified delimiter.

Syntax:

```
string.split(separator)
```

Example:

```
text = "apple,orange,banana"
fruits_list = text.split(',')
print(fruits_list)
```

6. `startswith()`

The `startswith()` method is used to check if a string starts with a specified prefix.

Syntax:

```
string.startswith(prefix)
```

Example:

```
text = " Programming"
starts_with_python = text.startswith("Python")
```

print(starts_with_python)

Output: True

7. `count()`

The `count()` method is used to count the occurrences of a substring in a string.

Syntax:

string.count(substring)

Example:

```
text = "programming is fun, programming is creative, programming is essential"
count_programming = text.count("programming")
print(count_programming)
```

Output: 3