**String(Operations and Functions)**Strings are a fundamental data type in Python used to represent text:

### **Basic String Operations:**

1. **Concatenation (**+**):**
   * Concatenate two strings to create a new string

str1 = "Hello"

str2 = " World"

result = str1 + str2

print(result)

1. **Repetition (**\***):**
   * Repeat a string multiple times.

str1 = "Python "

result = str1 \* 3

print(result)

1. **Indexing (**[]**):**
   * Access individual characters in a string using index notation.

my\_string = "Python"

print(my\_string[0])

1. **Slicing (**[:]**):**
   * Extract a portion of a string using slicing.

my\_string = "Python"

substring = my\_string[1:4]

print(substring) # Output: yth

1. **Length (**len()**):**
   * Get the length of a string.

pythonCopy code

my\_string = "Python"

length = len(my\_string)

print(length) # Output: 6

### String Methods:

upper() **and** lower()**:**

* + Convert a string to uppercase or lowercase.

my\_string = "Python"

upper\_case = my\_string.upper()

lower\_case = my\_string.lower()

print(upper\_case)

print(lower\_case)

1. strip()**:**
   * Remove leading and trailing whitespace from a string

my\_string = " Python "

stripped\_string = my\_string.strip()

print(stripped\_string) # Output: Python

1. replace()**:**
   * Replace a substring with another substring.

my\_string = "Hello, World!"

new\_string = my\_string.replace("World", "Python")

print(new\_string) # Output: Hello, Python!

1. split()**:**
   * Split a string into a list of substrings based on a specified delimiter.

pythonCopy code

my\_string = "apple,banana,orange"

fruits = my\_string.split(",")

print(fruits)

**A. Decision Making:**

Decision making statements can direct a program to execute on the based of instructions if a certain condition is True or another if it's False. Different decision making structures can be used such as **if statements**, **else-if statements,** and **nested if else statements**for decision making in python programming language.

## 1.The if statement

The if statement is used to execute a block of code only if a certain condition is True.

**syntax :**

**if condition:**

**# Code to be executed if the condition is True**

**# This code is indented under the if statement**

**Example:**

x = 10

if x > 5:

print("x is greater than 5")

print("This line is also part of the if block.")

### ****2.if-else statement:****

The if-else statement is used to execute one block of code if a condition is True and another block if the condition is False.

**Syntax:**

**if condition:**

**# Code to be executed if the condition is True**

**# This code is indented under the if statement**

**else:**

**# Code to be executed if the condition is False**

**# This code is indented under the else statement**

**Example:**

x = 3

if x > 5:

print("x is greater than 5")

else:

print("x is less than or equal to 5")

### ****3. if-elif-else statement:****

The if-elif-else statement allows checking multiple conditions and executing different blocks of code based on the first condition that is True.

**Syntax:**

**if condition1:**

**# Code to be executed if condition1 is True**

**# This code is indented under the if statement**

**elif condition2:**

**# Code to be executed if condition1 is False and condition2 is True**

**# This code is indented under the elif statement**

**elif condition3:**

**# Code to be executed if both condition1 and condition2 are False, and condition3 is True**

**# This code is indented under the elif statement**

**# You can have multiple elif blocks**

**else:**

**# Code to be executed if all conditions are False**

**# This code is indented under the else statement**

**Example:**

x = 10

if x > 10:

print("x is greater than 10")

elif x == 10:

print("x is equal to 10")

else:

print("x is less than 10")

**B. Looping Statement:**

A looping statement in programming is a construct that allows a block of code to be executed repeatedly, either a specific number of times or until a certain condition is met. Loops are essential for automating repetitive tasks and iterating over sequences or collections of data. In Python, there are two main types of looping statements: **for** loop and **while** loop.

**1. For Loop:**

* The for loop is used for iterating over a sequence (such as a list, tuple, string, or range).
* It allows you to execute a block of code for each item in the sequence.
* The loop variable takes on each value in the sequence during each iteration.

**Syntax:**

**for item in sequence:**

**# Code to be executed in each iteration**

**Example:**

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

for fruit in fruits:

print(fruit)

**2.While Loop**

* The while loop is used for executing a block of code as long as a specified condition is True.
* It continues iterating until the condition becomes False.
* The loop condition is evaluated before each iteration.

**Syntax:**

while condition:

# Code to be executed as long as the condition is True

# This code is indented under the while loop

**OR:**

intilisation

While condition

print statement

increment

**Example:**

num = 5

factorial = 1

counter = 1

while counter <= num:

factorial \*= counter

counter += 1

print(f"The factorial of {num} is {factorial}")

**Example:2**

count = 0

while count < 5:

print(count)

count += 1

**Example:3**

i=1

while(i<6)

print(i)

**C. Control Statement**

Control statements in programming are structures that manage the flow of execution in a program. They enable the program to make decisions, repeat a block of code multiple times, and control the overall flow of the program based on certain conditions. Control statements help in writing dynamic and flexible programs.

**1. Break:**

The **break** statement in Python is used to terminate the execution of a loop prematurely.

**Example:1**

Using for loop:

for i in range(10):

print(i)

if i == 2:

break

**Example:2**

Using While Loop

i = 0

while i < 6:

print(i)

if i == 4:

break

i += 1

**2 .Continue:**

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

**Using While Loop**

i = 1

while i < 7:

i += 1

if i == 5:

continue

print(i)

**Using Range**

for i in range(10):

i += 1

if i == 4:

continue

print(i)

**D. RANGE:**

In Python, range() is a built-in function used to generate a sequence of numbers. It's commonly used in for loops to iterate a specific number of times. The range() function can take one, two, or three arguments:

*****Syntax:*** *range(start, stop, step)***

1. range(stop): Generates numbers from 0 up to (but not including) the specified stop value.
2. range(start, stop): Generates numbers from the start value up to (but not including) the stop value.
3. range(start, stop, step): Generates numbers from the start value up to (but not including) the stop value with the specified step increment.

**Example:1**

for i in range(5):

print(i, end=" ")

print()

**1.Nested Loop:**

A nested loop in Python is a loop inside another loop. You can have a loop (inner loop) inside another loop (outer loop).

**Example:1**

for x in range(3):

for i in range(1,5):

print(i, end=””)

print()

**Eg:2**

for x in range(4):

for i in range(x)

print(“\*”, end=””)

print()

**E. Enumerate()**

Python enumerate() function returns an enumerated object. It takes two parameters first is a sequence of elements and the second is the start index of the sequence. We can get sequence elements either through a loop or next() method.

### **Syntax:**

### enumerate(sequence,start=0)

**Example**

fruits = ['apple', 'banana', 'cherry']

# The enumerate() function returns a tuple containing the index and value of each element

for i, fruit in enumerate(fruits):

print(i, fruit)

**F. Type Conversion/Casting**

One of the key features in Python is "Typecasting". Typecasting in Python allows programmers to convert variables or data from one data type to another. It enables smooth manipulation of data. Typecasting in Python can be very useful in dealing with large data sets where the data is present in different data types.  
**Example1:**

integer\_number = 123

float\_number = 1.23

new\_number = integer\_number + float\_number

# display new value and resulting data type

print("Value:",new\_number)

print("Data Type:",type(new\_number))