Day 3:

For loop in continuation:

1. This is how we iterate over sequences

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

for number in numbers:

print(number)

1. The range() function is often used to generate a sequence of numbers to iterate over.

for i in range(1, 6):

print(i)

1. We can also use \_ in place of “i” in above example

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

for index, fruit in enumerate(fruits):

print(f"Index: {index}, Fruit: {fruit}")

output:

Index: 0, Fruit: apple

Index: 1, Fruit: banana

Index: 2, Fruit: cherry

1. We can use a nested for loop:

for i in range(3):

for j in range(2):

print(i, j)

1. Continue:example: for i in range(1, 6):

if i == 3:

continue # Skip iteration when i is 3

print(i)

1. For iterating over strings

text = "Hello"

for char in text:

print(char)

1. For iterating over Dictionaries:

student\_grades = {'Alice': 95, 'Bob': 87, 'Charlie': 92}

for name, grade in student\_grades.items():

print(f"{name}: {grade}")

1. we also use for else loop :

for i in range(5):

print(i)

else:

print("Loop completed normally.")

**Tuple Unpacking in python:**

1. Tuple unpacking is also known as "tuple assignment" because it allows you to assign values from a tuple to multiple variables in a single statement.
2. Tuple unpacking is done using a comma-separated list of variables on the left-hand side of an assignment statement, and a tuple on the right-hand side.

a, b, c = (1, 2, 3)

1. The number of variables in the left side should match the number of elements in the right hand side otherwise VALUE ERROR will occur.

Ex=a,b=(1,2,3) wil give value error

1. You can use an underscore \_ to ignore specific elements of the tuple that you don't need. This is especially useful when you want to unpack only certain elements.

x, \_, z = (10, 20, 30)

1. To collect all the variables of tuple into a single variable we can use \* operator:

Ex: a, \*rest = (1, 2, 3, 4, 5)

1. We can Swap values using tuple unpacking:x=(1,2)

y=(3,4)

x,y=y,x

1. We can use tuple to store the function return values:

Tuple unpacking is commonly used when functions return multiple values as a tuple.

def get\_coordinates():

return (10, 20)

x, y = get\_coordinates()

here the value 10 is stored in x and 20 is stored in y

1. coordinates = [(1, 2), (3, 4), (5, 6)]

for x, y in coordinates:

print(f"X: {x}, Y: {y}")

**Functions:**

1. Functions are basically the block of code that we use to call them anywhere in the program.
2. We use def keyword to create a function.
3. The function body is indented and contains the code that the function executes when called. It can consist of one or more statements.
4. Local Variables: Variables defined inside a function are local to that function and have a limited scope. They are not accessible outside the function unless explicitly returned.
5. Global Variables: Variables defined outside of any function are considered global and can be accessed from within functions. However, it's a good practice to pass them as parameters rather than relying on global variables.
6. **Syntax:**

Def function\_name():

#code

**We use snake casing to name a function, that means we name it with the help of underscore.**

We can pass as many parameters as we want in a function.

We can pass string, list ,tuples,sets and dictionary are a function parameter

**return statement:** we use return keyword in a function .

help to store a function value in memory using any other variable

**example: def fun\_one(name):**

**print("name=",name)**

**fun\_one('shruti')# same indentation as def so it means this is outside the function**