Unit-3 Loops/iterative statements -Functions

Course Learning Objectives

- Students will come to know about loops statements
- Students can understand about the how while loop statement
- Students will learn about for loop statement

Introduction

Looping is a powerful programming technique through which a group of statements is executed repeatedly, until certain specified condition is satisfied. Looping is also called a repetitive or an iterative control statement.

A loop in a program essentially consists of two parts, one is called the body of the loop and other is known as a control statement. The control statement performs a logical test whose result is either **True or False**. If the result of this logical test is true, then the statements contained in the body of the loop are executed. Otherwise, the loop is terminated.

There must be a proper logical test condition in the control statement, so that the statements are executed repeatedly and the loop terminates gracefully. If the logical test condition is carelessly designed, then there may be possibility of formation of an infinite loop which keeps executing the statements over and over again.

Module-1

3.1 while loop

This is used to execute a set of statements repeatedly as long as the specified condition is true. It is an indefinite loop. In the while loop, the condition is tested before executing body of the statements. If the condition is True, only body of the block of statements will be executed otherwise if the condition is False, the control will jump to other statements which are outside of the while loop block.

Syntax:

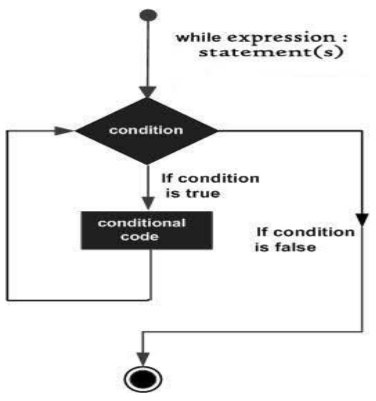
```
while(logexp):
          block of Statements
```

Where,

while → is a keyword
Logexp →is a logical expression that results in either **True or Fasle**
Statement→ may be simple or a compound statement

Here, first of all the logical expressions is evaluated and if the result is true (non-zero) then the statement is repeatedly executed. If the result is false (zero), then control comes out of the loop and continues with the next executable statements.

FLOWCHART:



In while loop there are mainly it contains three parts, which are as follows

1. Initialization: to set the initial value for the loop counter. The loop counter may be an increment counter or a decrement counter
2. Decision: an appropriate test condition to determine whether the loop be executed or not
3. Updation: incrementing or decrementing the counter value.

Example Programs:

Q1. Write a program to display your name up to 'n' times

```
n=int(input('Enter the number'))
#from the above line we take input from the user
i=1# assigning value to i variable
while(i<=n):# checking the condition
    print('RGUKT')#displaying output
    i=i+1#increasing i value here
print('good bye')
```

When you run the following program, then output will be display like

```
Enter the number
3
RGUKT
RGUKT
RGUKT
good bye
```

Q2. Write a program to display natural numbers up to n

```
n=int(input('Enter the number\n'))
i=1
print('Natural numbers are up to %d'%n)
while(i<=n):
    print(i)
    i=i+1
```

Output:

```
Enter the number
7
Natural numbers are up to 7
1
2
3
4
5
6
7
```

Q3.Write a program to display even numbers up to n

```
n=int(input('Enter the number: '))
i=1# assigning value 1 to i or intilization
print('Even numbers up to %d'%n)
while(i<=n):
    if(i%2==0):# here checking the condition whether the value is divisible or not
        print(i)# if it is divisible printing even numbers
    i=i+1# increment operator
```

Output:

```
Enter the number: 10
Even numbers up to 10
2
4
6
8
10
```

3.1.1 The infinite while loop: A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Program to demonstrate infinite loop:

```
while True:
    print('Hi')
print('Good bye')
```

```
x=1
while(x==1):
    print('Hi')
print('Good bye')
```

Above two programs that never stop, that executes until your Keyboard Interrupts (ctrl+c). Otherwise, it would have gone on unendingly. Many 'Hi' output lines will display when you executes above two programs and that never display 'Good bye'

3.1.2 While loop with else clause/statement: Python allows an optional else clause at the end of a while loop. This is a unique feature of Python. If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

But, the while loop can be terminated with a break statement. In such cases, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is False

Syntax:

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

Example Program:

Q1. Write a program to demonstrate while loop with else block.

```
i=1
while(i<=3):
    print('RGUKT')
    i=i+1
else:
    print('Good bye')
```

Output:

```
RGUKT
RGUKT
RGUKT
Good bye
```

Q2. Write a program to demonstrate else block with break statement

```
i=1
while(i<=6):
    if(i==4):
        break
    print(i)
    i=i+1
else:
    print('Never executes thie else block')
```

Output:

```
1
2
3
```

Module-2

3.2 for loop

Like the while loop, for loop works, to repeat statements until certain condition is True. The for loop in python is used to iterate over a sequence (list, tuple, string and range() function) or other iterable types. Iterating over a s sequence is called traversal. Here, by sequence we mean just an ordered collection of items. The for loop is usually known as definite loop because the programmer knows exactly how many times the loop will repeat.

Syntax:

```
for counter_variable in sequence:
        block of statements
```

Here counter_variable is the variable name that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for the loop is separated from the rest of the code using indentation.

Example programs:

Q7. Write a program to make sum of all numbers stored in a list.

```
#listof numbers
numbers=[5, 2, 7, 10, 14]
sum=0#assigning value to variable
for i in numbers:#iterate over the list
    sum=sum+i #making sum here
print('The sum is',sum)#displayiing output
```

When you run the program, Output will be:

```
The sum is 38
```

3.2.1 The range() function:

The range() function is a built-in function in python that is used to iterate over a sequence of numbers. The range() function contains three arguments/parameters like

range(start, end, step)

The range() generates a sequence of numbers starting with start(inclusive) and ending with one less than the number 'end'. The step argument is optional
By default, every number in the range is incremented by 1. Step can be either a positive or negative value but it cannot be equal to zero

Example Program:

Q8. Write a program to print first n natural numbers using a for loop

```
n=int(input('Enter the number: '))
for i in range(1,n,1):
    print(i,end=' ')
```

```
n=int(input('Enter the number: '))
for i in range(1,n):
    print(i,end=' ')
```

For both programs, output will be the same

```
Enter the number: 10
1 2 3 4 5 6 7 8 9
```

Q9. If you want to display including the given input number you can write a program like

```
n=int(input('Enter the number: '))
for i in range(1,n+1):
    print(i,end=' ')
```

Output:

```
Enter the number: 10
1 2 3 4 5 6 7 8 9 10
```

3.2.2 for loop with else:

A for loop can have an optional else block. The else part is executed when the loop has exhausted iterating the list. But, when the break statement use in for loop, the else part is ignored.
Hence, a for loop's else part runs if no break occurs.

Example:

Q. Program to demonstrate else block in for loop

```
for i in range(5):
    print(i)
else:
    print("No items left.")
```

Output:

```
0
1
2
3
4
No items left.
```

Here, the for loop prints numbers from the starting number to less than ending number. When the for loop exhausts, it executes the block of code in the else and prints No items left.

Q. Program to demonstrate else block in for loop with break statement:

```
for i in range(5):
    if(i==3):
        break
    print(i)
else:
    print('This else block not executes')
```

Output:

```
0
1
2
```

3.3 Nested loops

Python programming language allows using one loop inside another loop which is called a nested loop. Although this feature will work for both while loop as well as for loop. The syntax for a nested while loop statement in Python programming language is as follows

Syntax:

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
   statements(s)
```

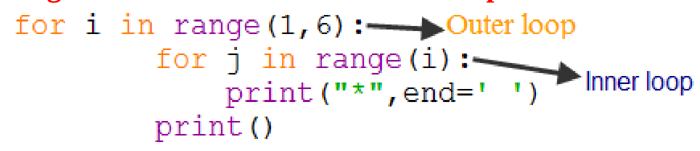
```
while expression:
   while expression:
      statement(s)
   statement(s)
```

Note:

You can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.

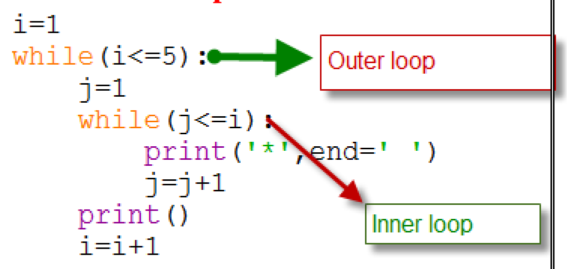
Example Program:

Program to demonstrate nested for loop



```
for i in range(1,6):
        for j in range(i):
            print("*",end=' ')
        print()
```

Nested while loop:



```
i=1
while(i<=5):
    j=1
    while(j<=i):
        print('*',end=' ')
        j=j+1
    print()
    i=i+1
```

Output:

If you run above two programs output will be the same

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

Module-3

3.4 Control statements

Loops iterate over a block of code until the test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression. These can be done by loop control statements. Loop control statements change execution from its normal sequence.

There are three different kinds of control statements which are as follows

1. break
2. continue
3. pass

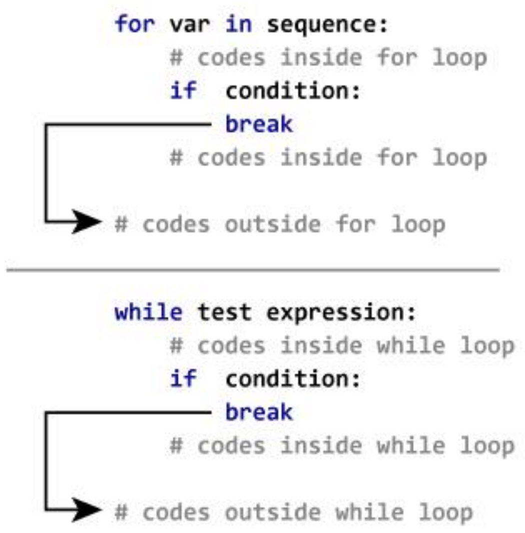
3.4.1 Break Statement: The break statement in Python terminates the current loop and resumes execution at the next statement (out of the loop). If the break statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.

This break statement is used in both for and while loops.

Syntax:

break

Flowchart



```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

Example Programs:

Write a program to demonstrate break statement by using for loop and while loop

for loop

```
for i in range(1, 11):
    if(i==5):
        break
    print(i)
print("Broke out of loop at i=",i)
```

when i equals 5 the loops break

while loop

```
i=1
while(i<11):
    if(i==5):
        break
    print(i)
    i=i+1
print("Broke out of loop at i=",i)
```

Output:

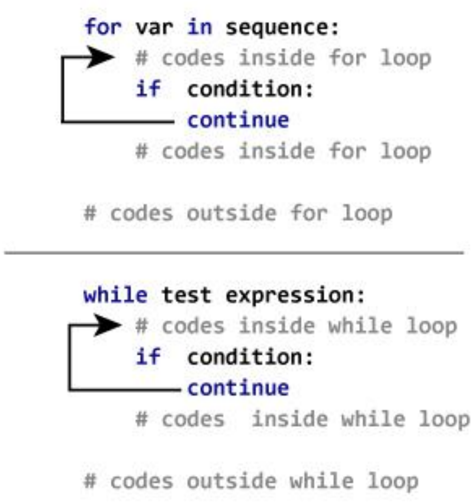
```
1
2
3
4
Broke out of loop at i= 5
```

3.4.2 Continue Statement: The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration. Continue statement is opposite to break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

Syntax:

continue

Flowchart:



```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

Example Programs:

Write a program to demonstrate continue statement by using for loop

```
for i in range(1, 10):
    if(i==5):
        continue
    print(i)
```

Output:

```
1 2 3 4 6 7 8 9
```

In output, 5 integer value is skipped based on the if condition and continues flow of the loop until the condition satisfied.

3.4.3 Pass Statement: The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is a null statement. However, nothing happens when the pass statement is executed. It results in no operation (NOP). Pass statement can also be used for writing empty loops, functions and classes.

Syntax:

```
pass
```

Example Program:

Program to demonstrate pass statement using for loop

```
for i in range(1, 10):
    if(i==5):
        pass
    print(i, end=' ')
```

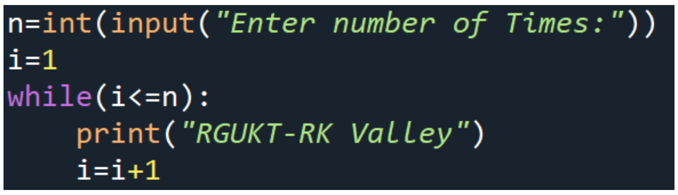
Output:

```
1 2 3 4 5 6 7 8 9
```

Solved Questions

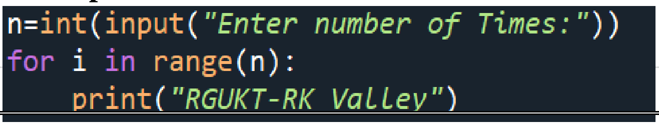
1. **Write a program to display your name up to n times.**

   Using while loop



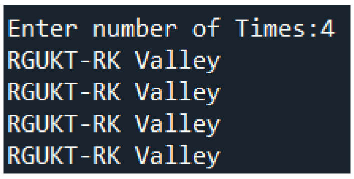
```
n=int(input("Enter number of Times:"))
i=1
while(i<=n):
    print("RGUKT-RK Valley")
    i=i+1
```

   Using for loop



```
n=int(input("Enter number of Times:"))
for i in range(n):
    print("RGUKT-RK Valley")
```

Output:



```
Enter number of Times:4
RGUKT-RK Valley
RGUKT-RK Valley
RGUKT-RK Valley
RGUKT-RK Valley
```

2. **Write a program to display sum of odd numbers up to n**

# Using while loop

```
n=int(input("Enter n Value:"))
i=1
s=0
while(i<=n):
   if(i%2!=0):
      s=s+1
   i=i+1
print("Sum of Odd Values", s)
```

# Using for loop

```
n=int(input("Enter n Value:"))
s=0
for i in range(1,n+1):
   if(i%2!=0):
      s=s+1
print("Sum of Odd Values", s)
```

3. **Write a program to display numbers of factors to the given number**

# Using while loop

```
n=int(input("Enter n Value:"))
i=1
c=0
while(i<=n):
   if(n%i==0):
      c=c+1
   i=i+1
print("count of factors is ", c)
```

# Using for loop

```
n=int(input("Enter n Value:"))
c=0
for i in range(1,n+1):
   if(n%i==0):
      c=c+1
print("count of factors is ", c)
```

4. **Write a program to find given number is prime number or not**

```
n=int(input("Enter n Value:"))
```

```
c=0
for i in range(1,n+1):
    if(n%i==0):
        c=c+1
if(c==2):
    print("given number is prime number")
else:
    print("given number is not prime number")
```

Descriptive Questions:

1. Explain about while loop and for loop?
2. Explain about loop control statements?
3. Explain about range function with different arguments?

Unsolved Questions:

1. Write a program to print all-natural numbers in reverse (from n to 1).
2. Write a program to print all even numbers up to n
3. Write a program to print sum of all odd numbers between two intravel given.
4. Write a program to print table of any number.
5. Write a program to enter any number and calculate sum of its digits.
6. Write a program to print all Prime numbers between 1 to n
7. Write a program to enter any number and display perfect numbers between 1 to n
8. Write a program to enter any number and find its first and last digit.
9. Write a program to display given number is a number palindrome or not
10. Write a program to print all alphabets from a to z.
11. Write a program to enter any number and check whether it is Armstrong number or not.
12. Write a program to print all Strong numbers between 1 to n.
13. Write a program to print Fibonacci series up to n terms.
14. Star pattern programs - Write a program to print the given star patterns.

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

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

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

15. Write a Python program to construct the following patterns, using a nested loop number

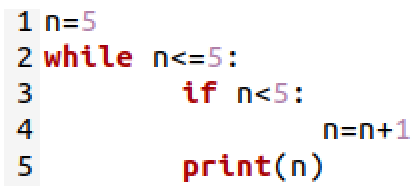
```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

```
P
PY
PYT
PYTH
PYTHO
PYTHON
```

Objective Questions:

1. A while loop in Python is used for what type of iteration?
   a. Indefinite
   b. Discriminate
   c. Definite
   d. Indeterminate
2. When does the else statement written after loop executes?
   a. When break statement is executed in the loop
   b. When loop condition becomes false
   c. Else statement is always executed
   d. None of the above
3. What do we put at the last of the for/while loop?
   a. Semicolon
   b. Colon
   c. Comma
   d. None of the above
4. Which of the following loop is work on the particular range in python?
   a. For loop
   b. While loop
   c. Do-while loop
   d. Recursion
5. How many times it will print the statement?, for i in range(100): print(i)
   a. 101
   b. 99
   c. 100
   d. 0

6. What is the result of executing the following code?



```
n=5
while n<=5:
        if n<5:
                n=n+1
        print(n)
```

   a. The program will loop indefinitely
   b. The value of number will be printed exactly 1 time
   c. The while loop will never get executed
   d. The value of number will be printed exactly 5 times

7. Which of the following sequences would be generated by given line of the code?
   a. 5 4 3 2 1 0 -1
   b. 5 4 3 2 1 0
   c. 5 3 1
   d. Error
8. Which of the following is a valid for loop in Python?
   a. for (i=0;i<=n;i++)
   b. for i in range(5)
   c. for i in range (1, 5):
   d. for i in range(0,5,1)
9. which statement is used to terminate the execution of the nearest enclosing loop in which it appears?

a. pass
b. break
c. continue
d. jump

10. Which statement indicates a NOP?
    a. pass
    b. break
    c. continue
    d. jump

Module 4 – Functions

3.5 Introduction:

A function is block of organized and reusable program code that performs a single or specific task is called function

A function is a set of instructions to carry out a particular task is called a function. Python enables its programmers to break up a program into number of smaller logical components/functions. The process of splitting the lengthy and complex programs into number of smaller units (sub programs) is called modularization (Advantages of modularization is, reusability of program, debugging is easier).

Functions are classified into two types, which are as follows...

1. Built-in-functions or Standard functions
2. User-defined functions

The Standard functions are also called library functions.

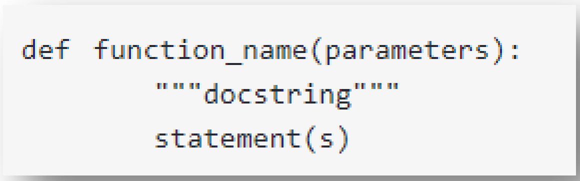
User-defined-function:

User can define function name to do a task relevant to their programs. Such functions are called user-defined functions.

Defining Function:

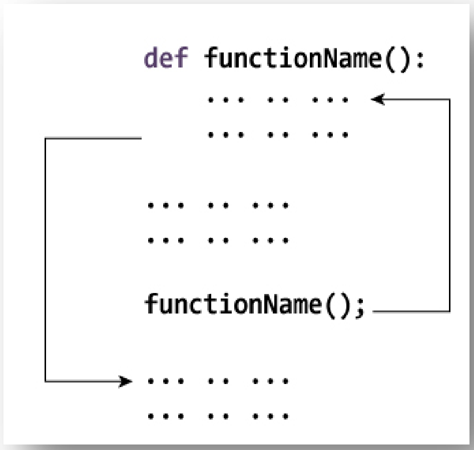
Function definition consists of a function header that identifies the function followed by the body of the function containing the executable code for that function.

Syntax:



```
def  function_name(parameters):
        """docstring"""
        statement(s)
```

How function works



```
def functionName():
    ... .. ...
    ... .. ...

... .. ...
... .. ...

functionName();

... .. ...
... .. ...
```

Above shown is a function definition which consists of following components.

1. Function blocks begin with the keyword **def** followed by the function name and parentheses ().
2. After the parentheses a colon (:) is placed.
3. Any input parameters or arguments should be placed within these parentheses.
4. The code block within the function is properly indented to form the block code.
5. A function may have a return [expression] statement. Return statement is optional. If it exits, it passes back an expression to the caller. A return statement with no arguments that returns None.

Example Program

```
def add(a,b):
    sum=a+b
    print("sum of two numbers",sum)
x=4
y=7
add(x,y)
```

Output

```
sum of two numbers 11
```

3.6 Function Call:

Defining a function means specifying its name, parameters that are expected and the set of instructions. Once the basic structure of a function is finalized, it can be executed by calling the function.

The function calls statement invokes the function. When a function is invoked, the program control jumps to the called function to execute the statements that are a part of the function. Once the called function is executed, the program control passes back to the calling function. The syntax of calling function that does not accept parameters is simply the name of the function followed by parenthesis, which is given as:

function_name ()

Function call statement has the following syntax when it accepts parameters.

function_name (variable1, Variable2, ...)

When the function is called, the interpreter checks that the correct number and type of arguments are used in the function call. It also checks the type of the returned value.

3.7 Function parameters:

A function can take parameters which are nothing but some values that are passed to it so that the function can calculate them to produce the desired results. These parameters are normal variables with a small difference that the values of these variables are defined (initialized) when we call the function and passed to the function.

Parameters are specified within the pair of parentheses in the function definition and are separated by commas. The function name and the number of arguments ain the function call must be same as that given in the function definition.

Types of arguments:

There are four types of arguments in python programming language. Those are as follows..

1. Required arguments
1. Keyword arguments
2. Default arguments
3. Variable-length arguments

Required arguments:

In the required arguments, the arguments are passed to a function in correct positional order. Also, the number of arguments in the function call should exactly match with the number of arguments specified in the function definition.

Example:

```
def rkv(x,y):
    add=x+y
    print add
a=3
b=7
rkv(a,b)
```

Output:

```
>>>
10
>>>
```

3.8 Keyword arguments:

Python programming allows functions to be called using keyword arguments in which the order (or position) of the arguments can be changed. The values are not assigning to arguments according to their position but based on their name (or keyword).

Keyword arguments when used in function calls, helps the function to identify the arguments by the parameter name.

Example: program that demonstrate keyword arguments

```
def fun(stry, intx, floaty):
    print "The string is:", stry
    print "The integer value is:", intx
    print "The floating point value is:",floaty

fun(floaty=9.99,stry="Rajiv Knowledge Valley", intx=7)
```

Output:

```
IDLE 2.6.4
>>>
The string is: Rajiv Knowledge Valley
The integer value is: 7
The floating point value is: 9.99
>>>
```

3.9 Default arguments:

Python allows users to specify function arguments that can have default values. It means that a function can be called with a few arguments than it is defined to have. That is, if the function accepts three parameters, but function call provides only two arguments, then the third parameter will be assigned the default (already specified) value.

The default value to an argument is provided by using the assignment operator (=). Users can specify a default value for one or more arguments.

Example:

```
def display(name, age, course="M.Tech"):
    print "Name:",name
    print "Age:",age
    print "course",course
display(course="B.Tech",name="Srujan", age=19)#keyword arguments
display(name="Aarahi",age=21)#default arguments for course
```

Output:

```
>>>
Name: Srujan
Age: 19
course B.Tech
Name: Aarahi
Age: 21
course M.Tech
>>>
```

3.10 Variable-length Arguments:

In some situation, it is not known in advance how many arguments will be passed to a function. In such cases, python allows programmers to make function calls with arbitrary number of arguments.

When we use arbitrary arguments or variable-length arguments, then the function definition uses an asterisk (*) before the parameter name.

Syntax:

```
def function_name(arg1,arg2, *var_agrs_tuple):
    "function statements"
    return [expression]
```

Example: Program to demonstrate the use of variable-length arguments.

```
def fun(name, *fav):
    print name,"Likes to read "
    for subject in fav:
        print subject,

fun("Saathvik","Physics","Chemistry","Telugu", "Java")
fun("Sindhu sha")
fun("Riyansh","mathematics", "DBMS","Perl","Network security")
```

Output:

```
>>>
Saathvik Likes to read
Physics
Chemistry
Telugu
Java
Sindhu sha Likes to read
Riyansh Likes to read
mathematics
DBMS
Perl
Network security
>>>
```