Python FOR LOOP

11.1.2.4 write programme code using a For loop;
11.1.2.5 define a range of values for a loop;
11.1.2.6 debug a program;
11.4.3.2 solve applied problems of various subject areas.

Python for loop

In Python, the for loop is used to iterate over a sequence such as a list, string, tuple, other iterable objects such as range.

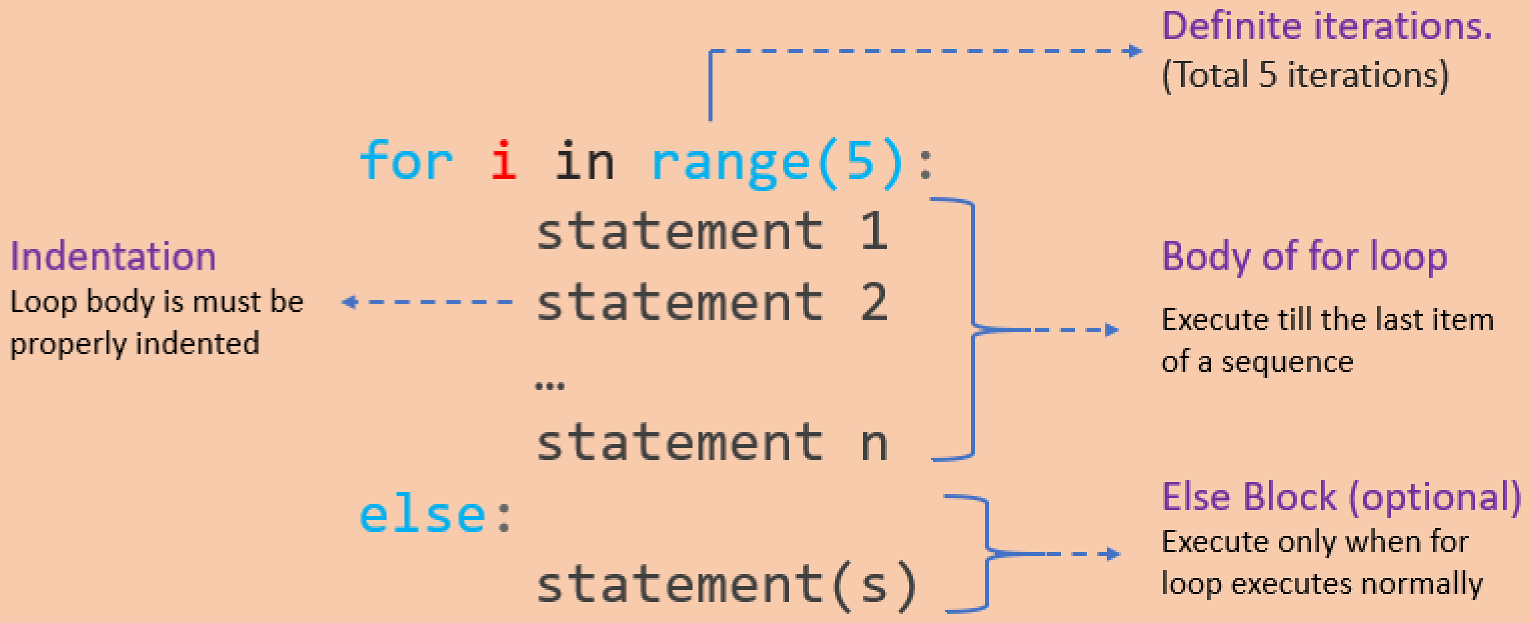
With the help of for loop, we can iterate over each item present in the sequence and executes the same set of operations for each item. Using a for loops in Python we can automate and repeat tasks in an efficient manner.

So the bottom line is using the for loop we can repeat the block of statements a fixed number of times. Let’s understand this with an example.

As opposed to while loops that execute until a condition is true, for loops are a fixed number of times, you need to know how many times to repeat the code.

Python for loop

A for loop is **used for iterating over a sequence and iterables** (like range, list, a tuple, a dictionary, a set, or a string).



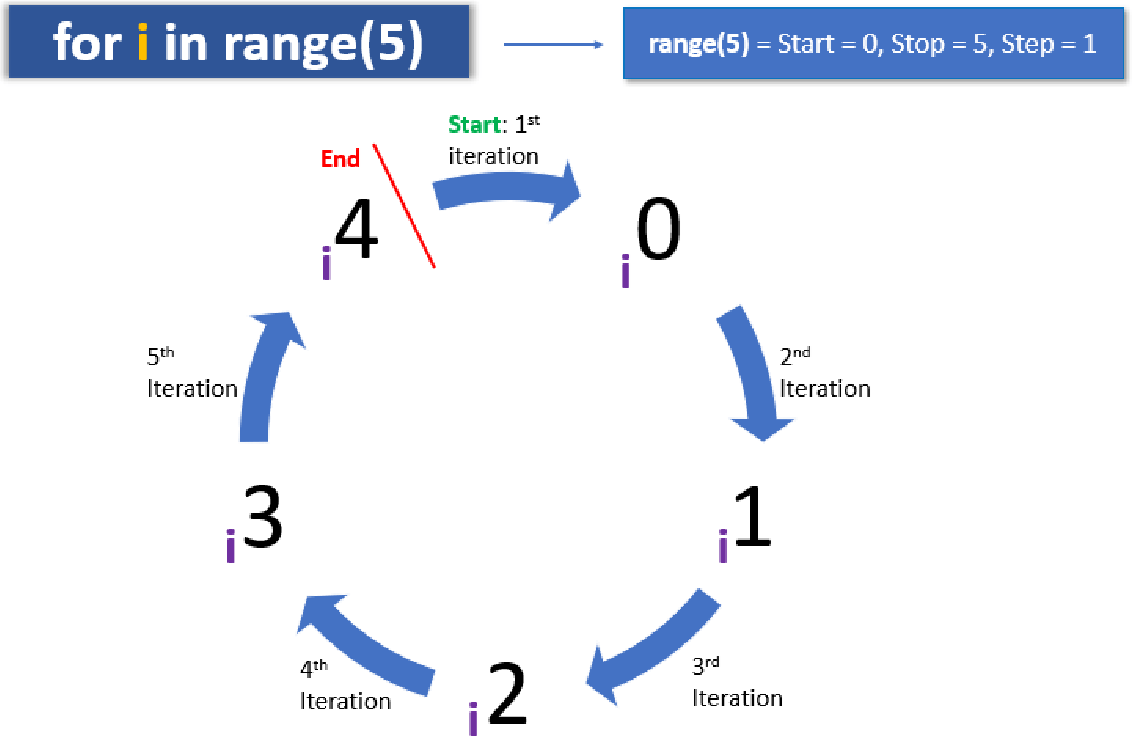
```
for i in range(5):
          statement 1
          statement 2
          ...
          statement n
else:
          statement(s)
```

Definite iterations.
(Total 5 iterations)
Indentation
Loop body is must be properly indented
Body of for loop
Execute till the last item of a sequence
Else Block (optional)
Execute only when for loop executes normally

for loop with range()

The range() function returns a sequence of numbers starting from 0 (by default) if the initial limit is not specified and it increments by 1 (by default) until a final limit is reached.

The range() function is used with a loop to specify the range (how many times) the code block will be executed. Let us see with an example.



How `for` loop works

The `for` loop is the easiest way to perform the same actions repeatedly. For example, you want to calculate the square of each number present in the list.

Write `for` loop to iterate a list, In each iteration, it will get the next number from a list, and inside the body of a loop, you can write the code to calculate the square of the current number.

Example: Calculate the square of each number of list

Python list is an ordered sequence of items. Assume you have a list of 10 numbers. Let's see how to want to calculate the square of each number using `for` loop.

```
numbers = [1, 2, 3, 4, 5]
# iterate over each element in list num
for i in numbers:
    # ** exponent operator
    square = i ** 2
    print("Square of:", i, "is:", square)
```

Output:

```
Square of: 1 is: 1
Square of: 2 is: 4
Square of: 3 is: 9
Square of: 4 is: 16
Square of: 5 is: 25
```

Why use `for` loop?

Let's see the use `for` loop in Python.

- **Definite Iteration**: When we know how many times we wanted to run a loop, then we use count-controlled loops such as for loops. It is also known as definite iteration. For example, Calculate the percentage of 50 students. here we know we need to iterate a loop 50 times (1 iteration for each student).
- **Reduces the code's complexity**: Loop repeats a specific block of code a fixed number of times. It reduces the repetition of lines of code, thus reducing the complexity of the code. Using `for` loops and `while` loops we can automate and repeat tasks in an efficient manner.
- **Loop through sequences**: used for iterating over lists, strings, tuples, dictionaries, etc., and perform various operations on it, based on the conditions specified by the user.

`If-else` in `for` loop

In this section, we will see how to use if-else statements with a loop. If-else is used when conditional iteration is needed. For example, print student names who got more than 80 percent.

The if-else statement checks the condition and if the condition is `True` it executes the block of code present inside the if block and if the condition is False, it will execute the block of code present inside the else block.

When the if-else condition is used inside the loop, the interpreter checks the if condition in each iteration, and the correct block gets executed depending on the result.

```
if condition:
    block of statements
else:
    block of statements
```

Example: Print all even and odd numbers

- In this program, `for` loop statement first iterates all the elements from 0 to 20.
- Next, The `if` statement checks whether the current number is even or not. If yes, it prints it. Else, the else block gets executed.

```python
for i in range(1, 11):
    if i % 2 == 0:
        print('Even Number:', i)
    else:
        print('Odd Number:', i)
```

Output:

```
Odd Number: 1
Even Number: 2
Odd Number: 3
Even Number: 4
Odd Number: 5
Even Number: 6
Odd Number: 7
Even Number: 8
Odd Number: 9
Even Number: 10
```

Loop Control Statements in for loop

Loop control statements change the normal flow of execution. It is used when you want to exit a loop or skip a part of the loop based on the given condition. It also knows as transfer statements.

Now, let us learn about the three types of loop control statements i.e., break, continue and pass.

Break for loop

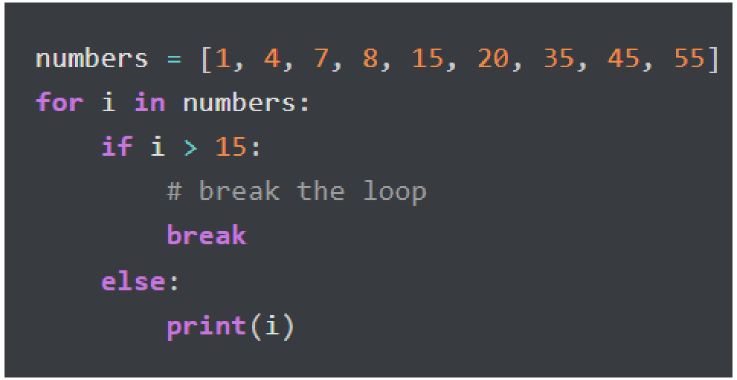
The break statement is used to terminate the loop. You can use the break statement whenever you want to stop the loop. Just you need to type the break inside the loop after the statement, after which you want to break the loop.

When the break statement is encountered, Python stops the current loop, and the control flow is transferred to the following line of code immediately following the loop.

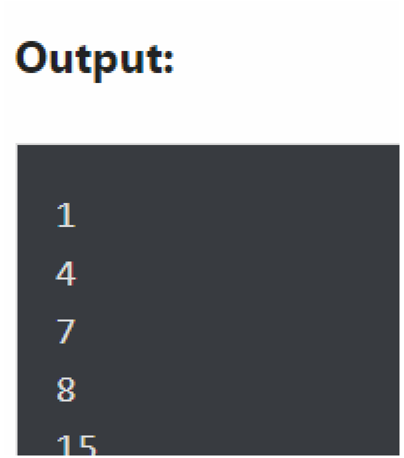
Example: break the loop if number a number is greater than 15

In this program, for loop iterates over each number from a list.

Next, the if statement checks if the current is greater than 15. If yes, then break the loop else print the current number



```python
numbers = [1, 4, 7, 8, 15, 20, 35, 45, 55]
for i in numbers:
    if i > 15:
        # break the loop
        break
    else:
        print(i)
```



Output:

```
1
4
7
8
15
```

Continue Statement in for loop

The continue statement skips the current iteration of a loop and immediately jumps to the next iteration

Use the continue statement when you want to jump to the next iteration of the loop immediately. In simple terms, when the interpreter found the continue statement inside the loop, it skips the remaining code and moves to the next iteration.

The continue statement skips a block of code in the loop for the current iteration only. It doesn't terminate the loop but continues in the next iteration ignoring the specified block of code. Let us see the usage of the continue statement with an example.

Example: Count the total number of 'm' in a given string.

- In this program, the `for` loop statement iterates over each letter of a given string.
- Next, the if statement checks the current character is m or not. If it is not m, it continues to the next iteration to check the following letter. else it increments the count

```
name = "mariya mennen"
count = 0
for char in name:
    if char != 'm':
        continue
    else:
        count = count + 1

print('Total number of m is:', count)
```

Output:

```
Total number of m is: 2
```

Note: In the case of an inner loop, it continues the inner loop only.

Pass Statement in for loop

The pass statement is a null statement, i.e., nothing happens when the statement is executed. Primarily it is used in empty functions or classes. When the interpreter finds a pass statement in the program, it returns no operation.

Sometimes there is a situation in programming where we need to define a syntactically empty block. We can define that block with the pass keyword.

Let us see the usage of the pass statement with an example.

```
num = [1, 4, 5, 3, 7, 8]
for i in num:
    # calculate multiplication in future if required
    pass
```

Iterate String using `for` loop

By looping through the string using `for` loop, we can do lots of string operations. Let's see how to perform various string operations using a `for` loop.

Example 1: Access all characters of a string

```
name = "Jessa"
for i in name:
    print(i, end=' ')
```

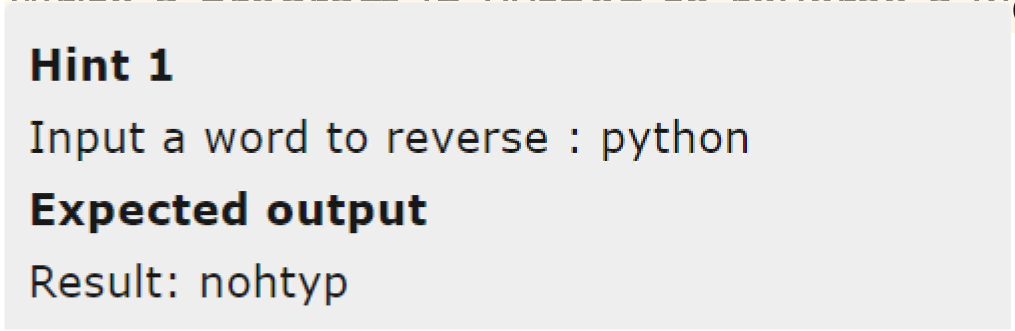
Run

Output:

```
J e s s a
```

TASK 1

Write a program in Python to reverse a word



Hint 1

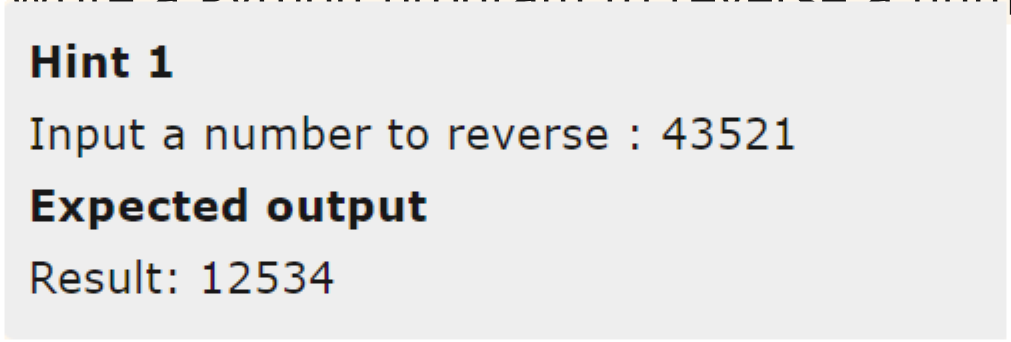
Input a word to reverse : python

Expected output

Result: nohtyp

TASK 2

Write a Python program to reverse a number.



Hint 1

```
Input a number to reverse : 43521
```

Expected output

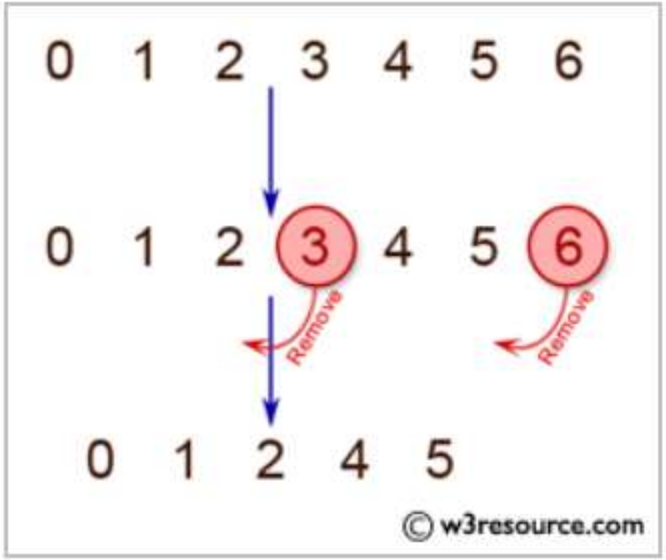
```
Result: 12534
```

TASK 3

Write a Python program that prints all the numbers from 0 to 6 except 3 and 6.

Note : Use 'continue' statement.

Pictorial Presentation:

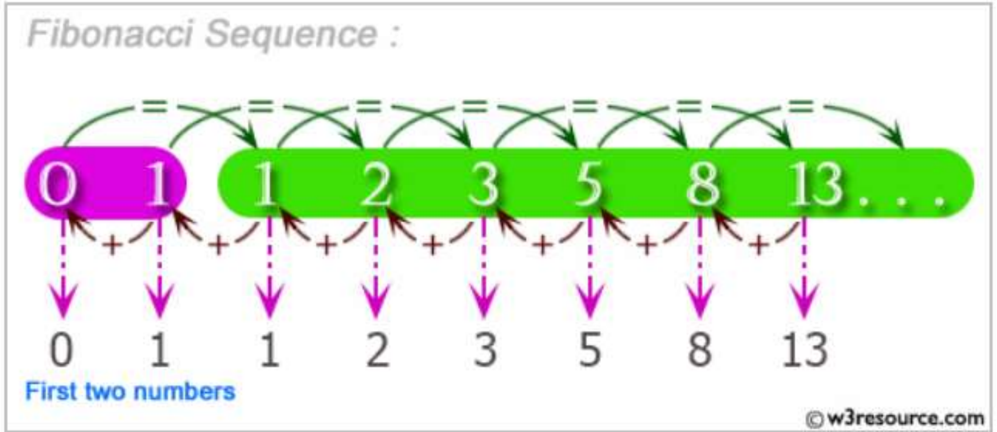


Task4

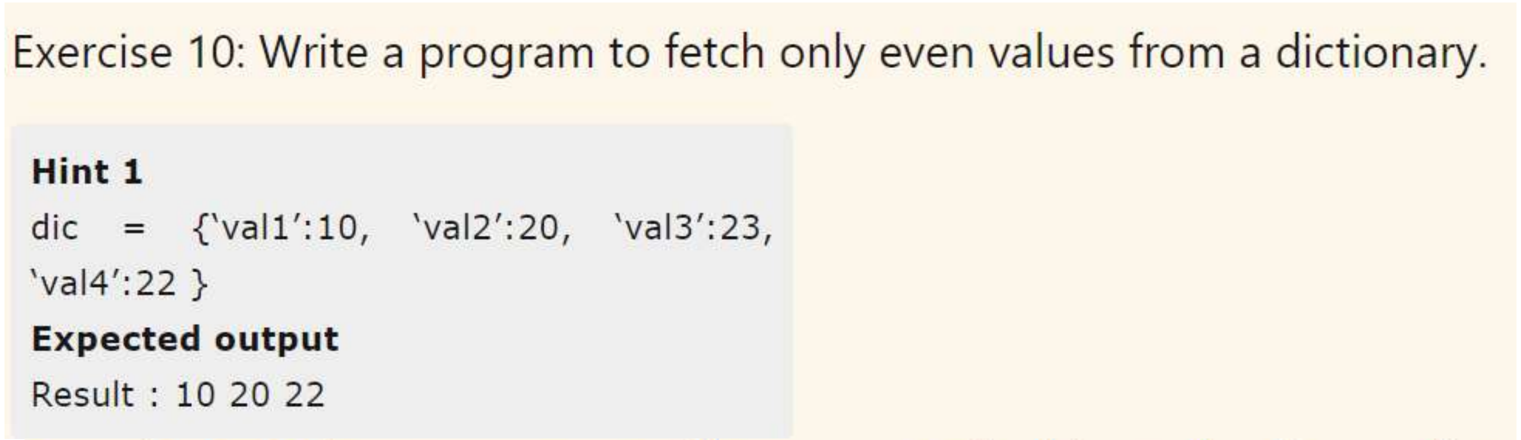
Write a Python program to get the Fibonacci series between 0 and 50.

Note : The Fibonacci Sequence is the series of numbers :
0, 1, 1, 2, 3, 5, 8, 13, 21, ...
Every next number is found by adding up the two numbers before it.

Pictorial Presentation:



for advanced



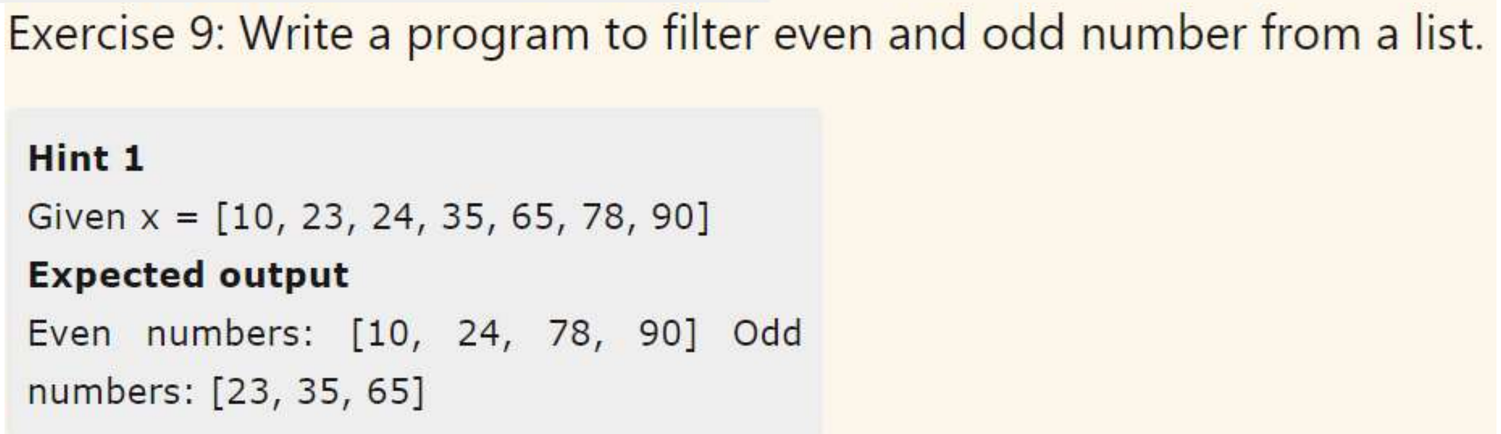
Exercise 10: Write a program to fetch only even values from a dictionary.

Hint 1

dic = {‘val1’:10, ‘val2’:20, ‘val3’:23, ‘val4’:22 }

Expected output

Result : 10 20 22



Exercise 9: Write a program to filter even and odd number from a list.

Hint 1

Given x = [10, 23, 24, 35, 65, 78, 90]

Expected output

Even numbers: [10, 24, 78, 90] Odd numbers: [23, 35, 65]

Exercise 8: Write a program that appends the type of elements from a list.

Hint 1

Given x = [23, ‘Python’, 23.98]

Expected output

Result:

[<class ‘int’>,<class ‘str’>,<class ‘float’>]

Exercise 7: WAP to separate positive and negative number from a list.

Hint 1

Given x = [23, 4, -6, 23, -9, 21, 3, -45, -8]

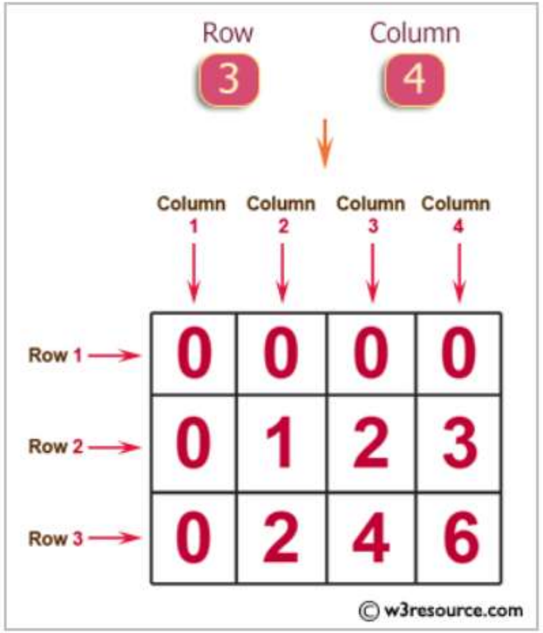
Expected output

Result:

Positive: [23, 4, 23, 21, 3] Negative: [-6, -45, -9, -8]

Write a Python program that takes two digits m (row) and n (column) as input and generates a two-dimensional array. The element value in the i-th row and j-th column of the array should be i*j.
Note :
i = 0,1.., m-1
j = 0,1, n-1.

Pictorial Presentation:


Row
3
Column
4
Column 1
Column 2
Column 3
Column 4
Row 1
Row 2
Row 3
0 0 0 0
0 1 2 3
0 2 4 6
© w3resource.com