



# **PROGRAMMING WITH PYTHON**

Karuna sheel

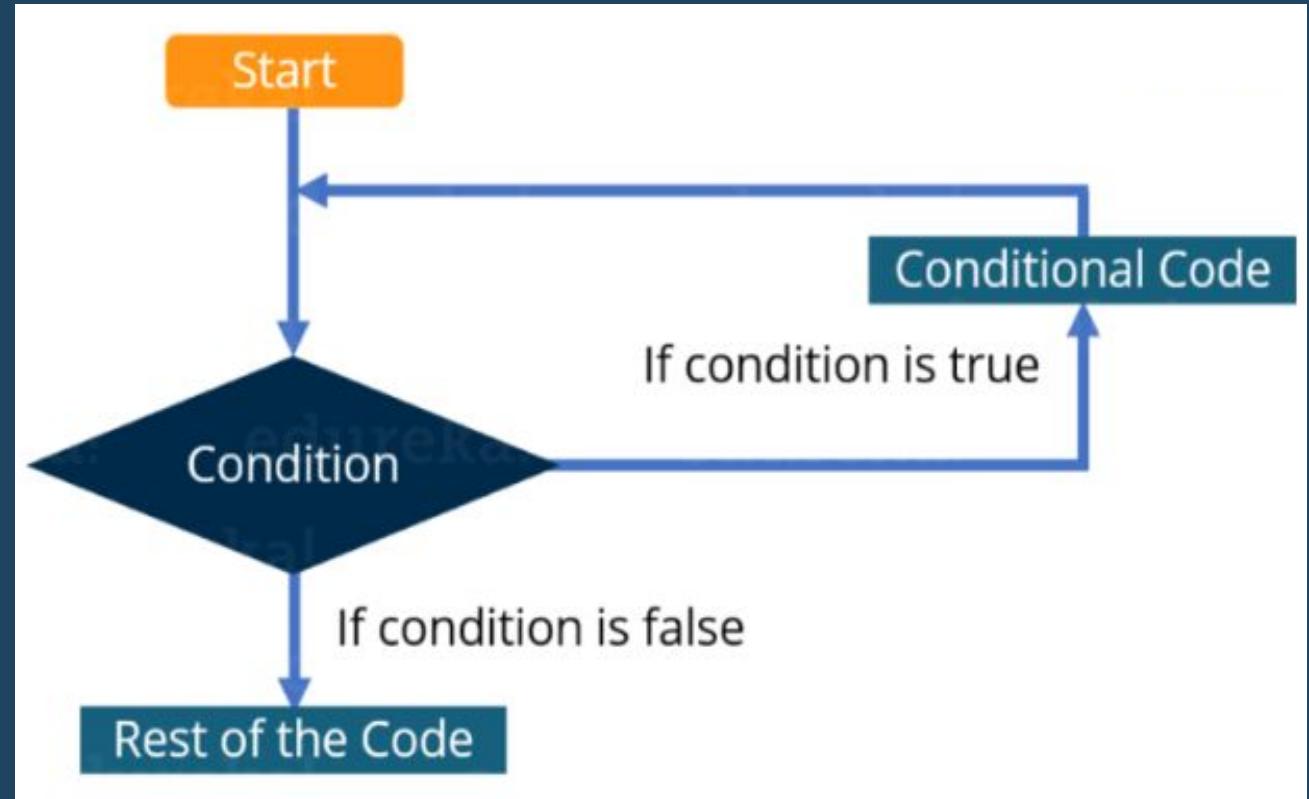
NIELIT kurukshetra

# PYTHON LOOPS

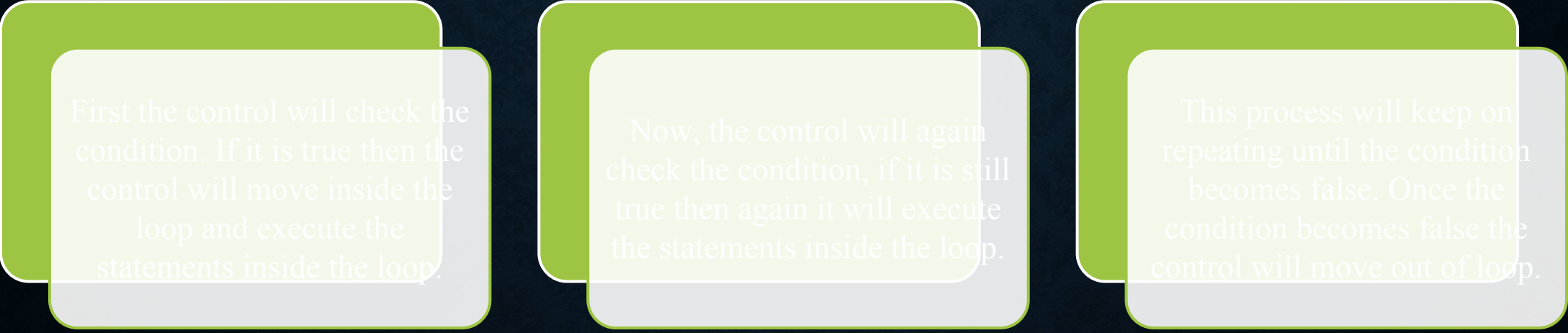
In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on

There may be a situation when you need to execute a block of code several number of times

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement:



# WORKING OF LOOP



First the control will check the condition. If it is true then the control will move inside the loop and execute the statements inside the loop.

Now, the control will again check the condition, if it is still true then again it will execute the statements inside the loop.

This process will keep on repeating until the condition becomes false. Once the condition becomes false the control will move out of loop.



# THERE ARE TWO TYPES OF LOOPS:

There is one more way to categorize loops:

Pre-test: In this type of loops the condition is first checked and then only the control moves inside the loop

Post-test: Here first the statements inside the loops are executed, and then the condition is checked

Python does not support Post-test loops.

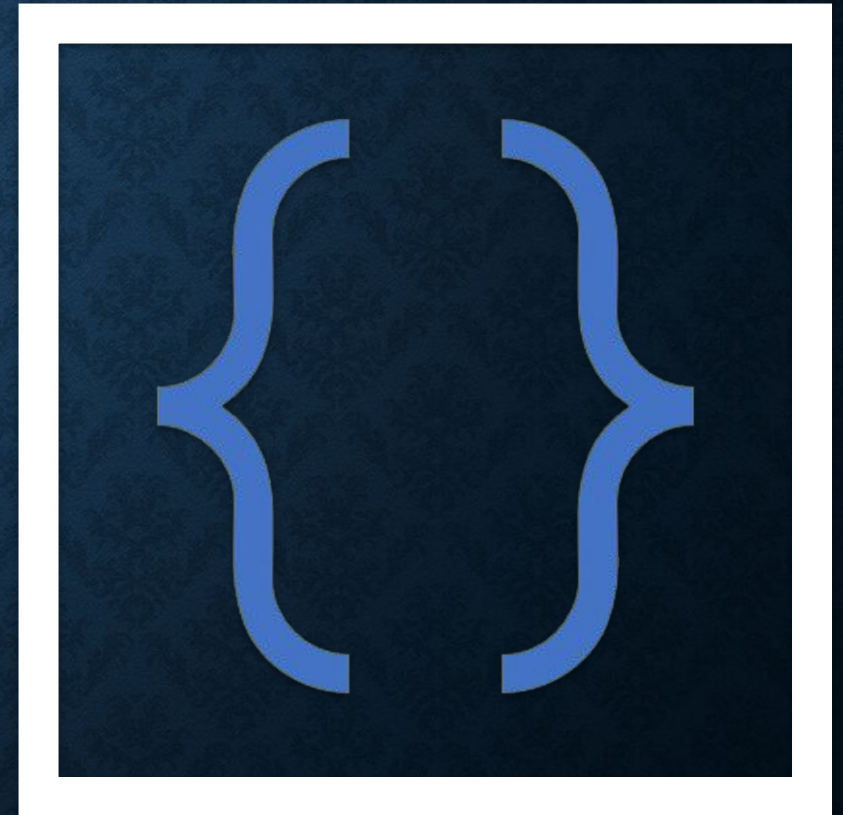
**Infinite:** When condition will never become false

**Finite:** At one point, the condition will become false and the control will move out of the loop

# ADVANTAGES OF LOOPS

There are the following advantages of loops in Python.

- ❑ It provides code re-usability.
- ❑ Using loops, we do not need to write the same code again and again.
- ❑ Using loops, we can traverse over the elements of data structures (array or linked lists).





# LOOPS IN PYTHON

In Python, there are three loops:

- While
- For
- Nested

## While Loop

Here, first the condition is checked and if it's true, control will move inside the loop and execute the statements inside the loop until the condition becomes false. We use this loop when we are not sure how many times we need to execute a group of statements or you can say that when we are unsure about the number of iterations.

Note: - Python doesn't have do-while loop.

### Syntax and Usage

```
count = 0
while (count < 10):
    print ( count )
    count = count + 1

print ("Good bye!")
```

### Output

```
0
1
2
3
4
5
6
7
8
9
Good bye!
```



# EXAMPLE WHILE LOOP

Print the table of number, entered by user-

```
i=1           #initialization
number=0
b=9
number = int(input("Enter the number?"))
while i<=10:   #Condition
    print("%d X %d = %d \n"%(number,i,number*i));
    i = i+1;    # increment/decrement
```



# EXAMPLE: PYTHON WHILE LOOP

```
# Program to add natural
# numbers up to
# sum = 1+2+3+...+n

# To take input from the user,
# n = int(input("Enter n: "))

n = 10

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

output will be:

Enter n: 10  
The sum is 55



# INFINITE WHILE LOOP

If the condition given in the while loop never becomes false then the while loop will never terminate and result into the infinite while loop.

Any non-zero value in the while loop indicates an always-true condition whereas 0 indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

## Example 1

**while (1):**

**print("Hi! we are inside the infinite while loop");**

Output:

**Hi! we are inside the infinite while loop  
(infinite times)**



## EXAMPLE 2

```
var = 1  
while var != 2:  
    i = int(input("Enter the number?"))  
    print ("Entered value is %d"%(i))
```

Output:

```
Enter the number?102  
Entered value is 102  
Enter the number?102  
Entered value is 102  
Enter the number?103  
Entered value is 103  
Enter the number?103  
(infinite loop)
```



# USING ELSE WITH PYTHON WHILE LOOP

Python enables us to use the else with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

Consider the following example.

```
i=1;
while i<=5:
    print(i)
    i=i+1;
else:print("The while loop exhausted");
```

**Output:**

**The while loop exhausted**





# For Loop

Like the While loop, the For loop also allows a code block to be repeated certain number of times. The difference is, in For loop we know the amount of iterations required unlike While loop, where iterations depends on the condition. You will get a better idea about the difference between the two by looking at the syntax:

Syntax

```
for variable in Sequence:  
    statements
```

Notice here, we have specified the range, that means we know the number of times the code block will be executed.

Example

```
i=1  
n=int(input("Enter the number up to which you want to print the natural numbers?"))
```

```
for i in range(0,10):  
    print(i,end = ' ')
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```



# EXAMPLES

```
i=1
n=int(input("Enter the number up to which you want to print the natural numbers?"))
for i in range(0,n):
    print(i,end = ' ')
```

Python for loop example : printing the table of the given number

```
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i));
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter:

range(2, 30, 3):

Example

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
    print(x)
```



# MORE ABOUT RANGE FUNCTION

We can generate a sequence of numbers using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as `range(start, stop, step_size)`. `step_size` defaults to 1 if not provided.

This function does not store all the values in memory; it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function `list()`.

## Example:

```
print(range(10))
```

```
print(list(range(10)))
```

```
print(list(range(2, 8)))
```

```
print(list(range(2, 20, 3)))
```

## Output:

```
range(0, 10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[2, 3, 4, 5, 6, 7]
```

```
[2, 5, 8, 11, 14, 17]
```



# EXAMPLE FOR LOOP WITH LIST

```
fruits = ['Banana', 'Apple', 'Grapes']
```

```
for index in range(len(fruits)):  
    print (fruits[index])
```

**Output: Banana      Apple      Grapes**



# ELSE IN FOR LOOP

A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.

The break keyword can be used to stop a for loop. In such cases, the else part is ignored.

Hence, a for loop's else part runs if no break occurs. Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

Example: 1

```
for x in range(6):
```

```
    print(x)
```

```
else:
```

```
    print("Finally finished!")
```

Output: 1

0

1

2

3

4

5

Finally finished!

Example: 2

```
digits = [0, 1, 5]
```

```
for i in digits:
```

```
    print(i)
```

```
else:
```

```
    print("No items left.")
```

Output: 2

0

1

5

No items left.



# SOME MORE EXAMPLES

```
# program to display student's marks from record
```

```
student_name = input("ENTER THE NAME YOU WANT TO SEARCH : ")
```

```
marks = {'James': 90, 'Sonu': 83, 'Jules': 55, 'Arthur': 77}
```

```
for student in marks:
```

```
    if student == student_name:
```

```
        print("Marks of ", student, "=", marks[student])
```

```
        break
```

```
else:
```

```
    print('No entry with that name found.')
```

Output

No entry with that name found.



# NESTED LOOPS

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example: Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]  
fruits = ["apple", "watermelon", "cherry"]
```

```
for x in adj:  
    for y in fruits:  
        print(x, y)
```

## Output:

```
red apple  
red banana  
red cherry  
big apple  
big banana  
big cherry  
tasty apple  
tasty banana  
tasty cherry
```