

Conditionals: Conditional statement in Python.

When we write a program, the statements in the program are executed one by one. This type of execution is called 'sequential execution'.

Flow Control Statement

Decision/Selection control statement

if
if else
if elif else

if Statement

if condition:
statements

First the condition is tested. If the condition is True, then the statements given after : are executed.

```
num = 1
if num == 1:
    print("one")
```

o/p: One

if-else statement

The format:

```
if condition:
    statements1
else:
    statements2
```

The if...else statement executes a group of statements when a condition is True, otherwise it will execute another group of statements.

```
x = 10
if x % 2 == 0:
    print("EVEN")
else:
    print("ODD")
```

II

Conditionals: Conditional statement in Python (if-else statement, its working and execution), Nested-if statement and Elif statement in Python, Expression Evaluation & Float Representation.

Loops: Purpose and working of loops, While loop including its working, For Loop, Nested Loops, Break and Continue.

Iterative/Loop statement

for
while

Nested-if statement and elif statement

When we test multiple conditions, then we use it.

```
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
else:
    statements4
```

```
x = int(input("Enter a number:"))
if (x > 0):
    print("Positive")
elif (x < 0):
    print("Negative")
else:
    print("Zero")
```

When condition 1 is True, statement 1 is executed.

When condition 2 is True, statement 2 is executed.

When condition 3 is True, statement 3 is executed.

When condition 3 is False, statement 4 will be executed.

which means statement 4 will be executed only if none of the conditions are True.

Expression Evaluation

>>> 2 + 2

4

In Python $2+2$ is called an expression.

Expressions consist of values and operators.

Expressions always evaluate i.e. they always reduce down to a single value.

Single value itself is an expression:

>>> 9

9

i.e. expression evaluates to itself

Expression evaluation is done by
Operator precedence.

U-1

>>> (5-1) * ((7+1) / (3-1))

16.0

4 * ((7+1) / (3-1))

4 * (8 / (3-1))

4 * (8 / 2)

4 * 4.0

16.0

no matter how complicated an expression is, All expressions are just values and operators, and it always evaluates down to a single value

>>> 2 + 3 * 6

20

>>> (2+3) * 6

30

Order of operations:

- 1) Parenthesis
- 2) Multiplication/Division
- 3) Addition/Subtraction

>>> 'a' + 'b'

'ab'

>>> 'a' * 3

'aaa'

Float Representation: The numbers having . (point) are float

>>> 4.0

4.0

>>> 1/3

0.3333333333333333

Python only prints a decimal approximation to the true decimal value of the binary approximation stored by machine. which means that Python keeps the number of digits manageable by displaying a rounded value instead

>>> 1/10

0.1

1/10 → true decimal value

0.1000000000000000055511151231257827021181583404541015625

Loops: while loops are also called as repetitive control statement or iterative control statement.
 ↓
 iterate statements till some condition is True, which means we use indexing here.
 o/p: abc
 abc ← $\begin{cases} x=1 \\ \text{while } x \leq 2: \\ \quad \text{print}("abc") \\ \quad x+=1 \end{cases}$ ($x+=1$ is same as $x=x+1$)

for
 ↓
 iterate statements for number of elements in a sequence. list, str, range, tuple etc.
 Number of iteration is known in advance.
 The for loop does not require an indexing variable to set beforehand, as the command itself allows for this.

for element in sequence:
 perform-action

$a = [1, 2, 3]$
 for i in a:
 print(i)

o/p: 1
 2
 3

Nested Loops
 It is possible to write one loop inside another loop.
 eg. we can write a for loop inside a while loop or a for loop inside another for loop. Such loops are called 'nested loops'.

for i in range(3):
 for j in range(4):
 print(i, '\t', j)

$i=0$ → for j in range(4):
 print(i, '\t', j)

$i=1$ → for j in range(4):
 print(i, '\t', j)

$i=2$ → for j in range(4):
 print(i, '\t', j)

o/p:
 0 0 0 0
 1 1 1 1
 2 2 2 2
 3 3 3 3

integer \rightarrow `range` \rightarrow integer sequence

```
for i in range(3):  
    print(i)
```

o/p:
0
1
2

break statement: can be used inside a for loop or while loop to come out of the loop. When break is executed, the Python interpreter jumps out of the loop to process the next statement in the program.

```
Statement 1  $\rightarrow$  x = 5  
Statement 2  $\rightarrow$  while x >= 1:  
    print(x)  
    x = x - 1  
    if x == 3:  
        break  
Statement 3  $\rightarrow$  print("out of loop")
```

o/p:
5
4
out of loop

The range function can generate a sequence of
eg `range(3)` \rightarrow (0, 3) Integers starting from 0

Continue statement: is used in a loop to go back to the beginning of the loop. Which means when continue is executed, the next repetition will start. Which also means that the statements written below continue are not executed.

```
x = 0  
while x < 5:  
    x = x + 1  
    if x > 3:  
        continue  
    print(x)  
print("out of loop")
```

continue next iteration of while loop

o/p:
1
2
3
out of loop