



# Django for Web Development & Artificial Intelligence

## Lecture - 03

### **Instructor:**

Md. Abu Noman Basar  
Django Developer



## Python Operators

The operator can be defined as a symbol that is responsible for a particular operation between two operands.

Python provides a variety of operators, which are described as follows:

- ☐ Arithmetic operators
- ☐ Assignment Operators
- ☐ Comparison operators
- ☐ Logical Operators
- ☐ Bitwise Operators
- ☐ Membership Operators
- ☐ Identity Operators

## Python Operators

### □ Arithmetic operators

Operators	Meaning	Example	Result
+	Addition	$4 + 2$	6
-	Subtraction	$4 - 2$	2
*	Multiplication	$4 * 2$	8
/	Division	$4 / 2$	2
%	Modulus operator to get remainder in integer division	$5 \% 2$	1
**	Exponent	$5 ** 2 = 5^2$	25
//	Integer Division/ Floor Division	$5 // 2$ $-5 // 2$	2 -3

# Python Operators

## □ Assignment operators

Operator	Name	Example
=	Assignment	$x = y$
+=	Add AND assignment	$x += y$ is same as $x = x + y$
-=	Subtract AND assignment	$x -= y$ is same as $x = x - y$
*=	Multiply AND assignment	$x *= y$ is same as $x = x * y$
/=	Divide AND assignment	$x /= y$ is same as $x = x / y$
%=	Modulus AND assignment	$x \% y$ is same as $x = x \% y$

## Python Operators

### □ Comparison operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

## Python Operators

### □ Logical operators

Operator	Name	Example
and	Logical and operator	x and y is true, if both x and y are true
or	Logical or operator	x or y is true, if either x or y is true
not	Logical not Operator	Returns false, if the result is true.

## Python Operators

### □ Bitwise operators

Operator	Name	Example	Result
&	Bitwise AND	6 & 3	2
	Bitwise OR	10   10	10
^	Bitwise XOR	2 ^ 2	0
~	Bitwise complement	~9	-10
<<	Left-Shift	10 << 2	40
>>	Right-Shift	10 >> 2	2

## Python Operators

### □ Bitwise complement(~) operators

**Ex:** ~12  
12 → 00001100  
↓ Reverse the binary  
-13 → 11110011

### □ Bitwise and (&) operators

**Ex:** 12 → 00001100  
13 → 00001101  
12 → 00001100

### □ Bitwise or( | ) operators

**Ex:** 12 → 00001100  
13 → 00001101  
13 → 00001101

### □ Bitwise XOR (^) operators

**Ex:** 12 → 00001100  
13 → 00001101  
1 → 00000001

### □ Left shift (<<) operators

**Ex:** 12 → 00001100  
00001100 ▪ 00  
0000110000

### □ Right shift ( ) operators

**Ex:** 12 → 00001100  
000011 ▪ 00  
000011



## Python Operators

### ❑ Membership operators

Operator	Name	Example
in	TRUE, if variable is in the list, string, dictionary, etc.	x in y
not in	TRU, if variable is not in the list, string, dictionary, etc.	x not in y

## Python Operators

### □ Identity operators

Operator	Name	Example
is	TRUE, if both the variable points to the same object, with same memory locations.	x is y
is not	TRUE, if both the variable points to different objects.	x is not y

## Python Conditions

### □ if, else statements

Decision-making is the most important aspect of almost all programming languages.



if



else



# Python Conditions

- if, elif, else statements



**if**



**elif**



**elif**

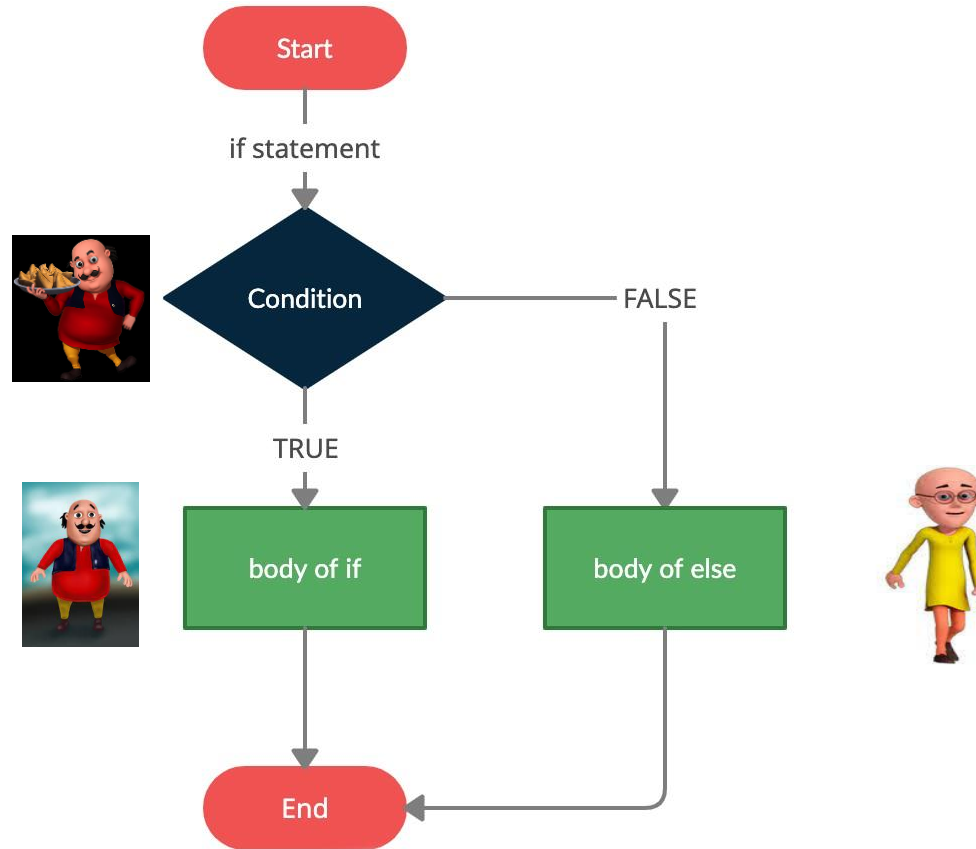


**else**



# Python Conditions

## □ if, else statements





# Python Conditions

## □ Python conditions

Python supports the usual logical conditions from mathematics.

- Equals: **`a == b`**
- Not Equals: **`a != b`**
- Less than: **`a < b`**
- Less than or equal to: **`a <= b`**
- Greater than: **`a > b`**
- Greater than or equal to: **`a >= b`**

## Python Conditions

### □ if, else statements

```
1 motu = 100
2 patlu = 50
3
4 if (motu>patlu):
5     print('Motu has more than Patlu')
6
7 else:
8     print('Patlu has more than Motu.')
```

## Python Conditions

□ if, elif, else statements

```
1 motu = 500
2 patlu = 100
3 jhotka = 1500
4
5 if (motu>patlu and motu>jhotka):
6     print('Motu have more than Patlu & Jhotka')
7
8 elif (patlu>motu and patlu>jhotka):
9     print('Patlu have more than Motu & Jhotka.')
10
11 else:
12     print('jhotka have more than motu & patlu')
```





## Python Loops

We can run a single statement or set of statements repeatedly using a loop command.

Python has two primitive loop commands:

- For loop
- while loop

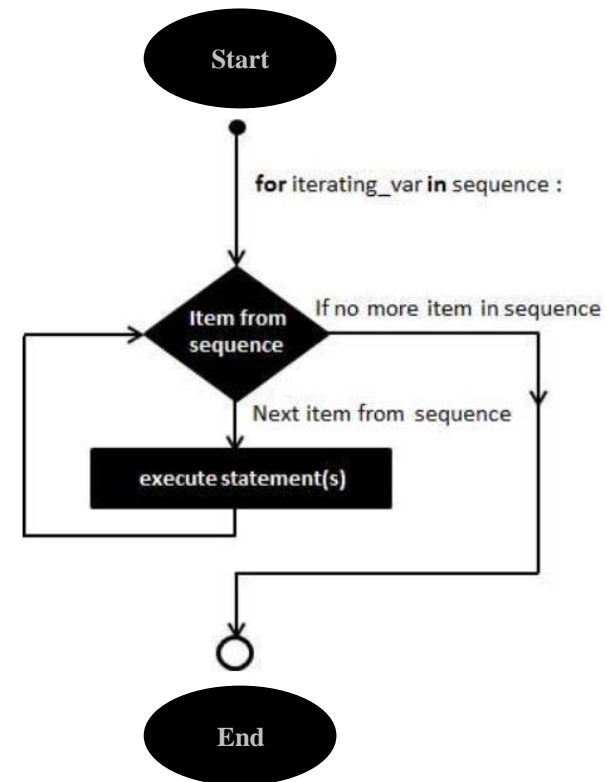
## Python For Loops

### □ For loop

A **for** loop is used for **iterating over a sequence** (that is either a **list**, a **tuple**, a **dictionary**, a **set**, or a **string**).

### Syntax

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





## Python For Loops

### ☐ break Statement

With the **break** statement, we can stop the loop before it has looped through all the items.

### ☐ continue Statement

With the **continue** statement, we can stop the current iteration of the loop, and continue with the next.



## Python For Loops

### ☐ **range() Function**

#### ☐ **range(stop)**

When you pass only one argument to the `range()` , it will generate a sequence of integers starting from 0 to stop -1

#### ☐ **range(start, stop)**

When you pass two arguments to the `range()`, it will generate integers starting from the start number to stop -1

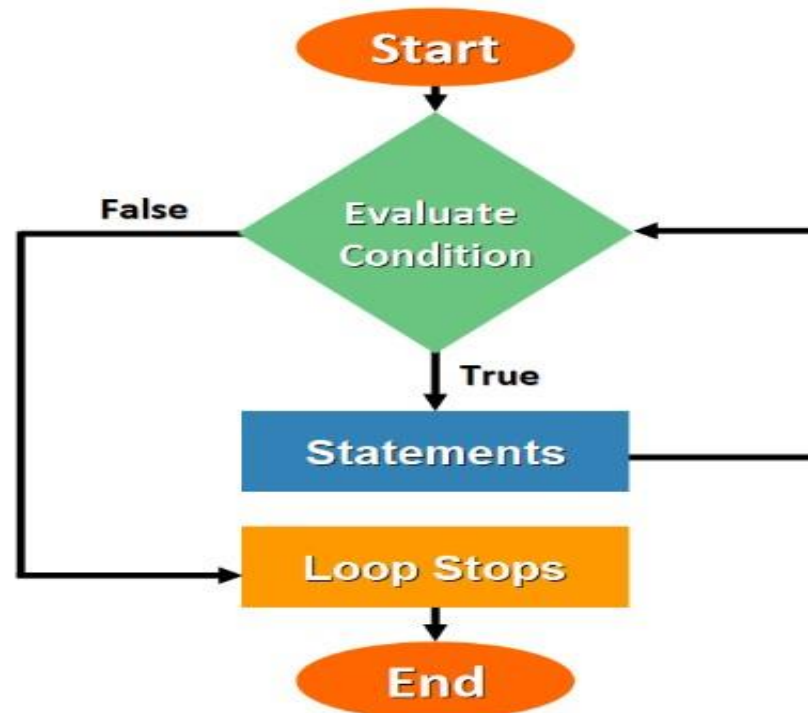
#### ☐ **range(start, stop, step)**

When you pass all three arguments to the `range()`, it will return a sequence of numbers, starting from the start number, increments by step number, and stops before a stop number.

## Python While Loops

### □ While loop

While Loop is used to execute a block of statements repeatedly until a given **condition is satisfied**. And when the condition becomes false, the line immediately after the loop in the program is executed.





## Python List

### □ List

A list is a collection of things, enclosed in [ ] and separated by a comma (,).

**Ex:** `course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]`

### □ len() Function

**Ex:** `course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]  
print(len(course))`

### □ Access List Items

**Ex:** `course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]  
print('List item: ', course[4])`

## Python List

### ❑ Range of indexes

**Ex:**

```
course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]
print('Range: ', course[2:5])
```

### ❑ Range of negative indexes

**Ex:**

```
course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]
print('Negative range: ', course[-6:-1])
```

### ❑ Change item value

**Ex:**

```
course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]
course[1] = 3
print('New list: ', course)
```

## Python List

### ❑ insert() method

- To insert a list item at a specified index, use the **insert()** method.

**Ex:**

```
course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]
course.insert(0, "AiQuest")
print('New list: ', course)
```

### ❑ append() method

- To add an item to the end of the list, use the **append()** method.

**Ex:**

```
course = ['python', 4, 'Django', 16, 'ML', 20, 'DL', 22]
course.append("StudyMart")
print('Append: ', course)
```



## Python List

### ❑ remove() method

- The **remove()** method removes the specified item.

**Ex:**

```
course = ['python',4, 'Django', 16, 'ML', 20,'DL',22]
course.remove(22)
print('Remove: ', course)
```

### ❑ pop() method

- The **pop()** method removes the specified index.

**Ex:**

```
course = ['python',4, 'Django', 16, 'ML', 20,'DL',22]
course.pop(6)
print('pop: ', course)
```

***\*\*\*If you do not specify the index, the pop() method removes the last item.***

## Python List

### ❑ del keyword

- The **del** keyword also removes the specified index.

**Ex:**

```
course = ['python',4, 'Django', 16, 'ML', 20,'DL',22]
del course[5]
print('del: ', course)
```

### ❑ clear() method

- The **clear()** method empties the list. The list still remains, but it has no content.

**Ex:**

```
course = ['python',4, 'Django', 16, 'ML', 20,'DL',22]
course.clear()
print('Remove list item : ', course)
```

## Python List

### ❑ `sort()` method

- List objects have a `sort()` method that will sort the list alphanumerically, ascending, by default.

**Ex:**

```
alphabetic = ['Python', 'Django', 'ML']  
alphabetic.sort()  
print('Alphabetic ascending order: ', alphabetic)
```

### ❑ `sort()` method (descending)

- To sort descending, use the keyword argument, `reverse = True`

**Ex:**

```
numeric = [34, 1, 20, 22, 5, 97, 11]  
numeric.sort(reverse=True)  
print('Numeric descending order: ', numeric)
```