codingninjas

Operators and Conditionals

Assignment Operators in Python

1. Assignment Operator: **“=”**

Description: Assigns a value to a variable.

Example: x = 5

2. Addition Assignment: **“+=”**

Description: Adds the value on the right to the variable and assigns the result to the variable.

Example : `x += 3` is equivalent to `x = x + 3`

3. Subtraction Assignment: **“-=”**

Description: Subtracts the value on the right from the variable and assigns the result to the variable.

Example : `x -= 3` is equivalent to `x = x - 3`

4. Multiplication Assignment: **“*=”**

Description : Multiplies the variable by the value on the right and assigns the result to the variable.

Example: `x *= 3` is equivalent to `x = x * 3`

5. Division Assignment: **“/=”**

Description: Divides the variable by the value on the right and assigns the result to the variable.
Example: `x /= 3` is equivalent to `x = x / 3`

6. Modulus Assignment: **“%=”**

Description: Computes the modulus of the variable with the value on the right and assigns the result to the variable.

Example: `x %= 3` is equivalent to `x = x % 3`

7. Floor Division Assignment: "//="

Description: Performs floor division on the variable with the value on the right and assigns the result to the variable.

Example : `x //= 3` is equivalent to `x = x // 3`

8. Exponentiation Assignment: "**="

Description: Raises the variable to the power of the value on the right and assigns the result to the variable.

Example : `x **= 3` is equivalent to `x = x ** 3`

9. Bitwise AND Assignment: "&="

Description: Performs a bitwise AND operation between the variable and the value on the right and assigns the result to the variable.

Example : `x &= 3` is equivalent to `x = x & 3`

10. Bitwise OR Assignment: "|="

Description: Performs a bitwise OR operation between the variable and the value on the right and assigns the result to the variable.

Example: `x |= 3` is equivalent to `x = x | 3`

11. Bitwise XOR Assignment: "^="

Description: Performs a bitwise XOR operation between the variable and the value on the right and assigns the result to the variable.

Example : `x ^= 3` is equivalent to `x = x ^ 3`

12. Bitwise Left Shift Assignment: **“<<=”**

Description: Performs a bitwise left shift operation on the variable by the value on the right and assigns the result to the variable.

Example: `x <<= 3` is equivalent to `x = x << 3`

13. Bitwise Right Shift Assignment: **“>>=”**

Description: Performs a bitwise right shift operation on the variable by the value on the right and assigns the result to the variable.

Example : `x >>= 3` is equivalent to `x = x >> 3`

Comparison Operators in Python

The operators that compare their operands' values are called comparison/ relational operators. Python has six most common relational operators. Let X and Y be the two operands and X = 5 and Y = 10.

Operator	**Description**	**Example**
==	If the values of two operands are equal, then the condition is **true**, otherwise, it is **false**. *Common Mistake:- Do not confuse it with the Assignment Operator(=).*	(X == Y) is **false**
!=	If the values of the two operands are not equal, then the condition is **true.**	(X != Y) is **true.**

>	If the value of the left operand is greater than the value of the right operand, then the condition is **true.**	(X > Y) is **false**
<	If the value of the left operand is less than the value of the right operand, then the condition is **true.**	(X < Y) is **true.**
>=	If the value of the left operand is greater than or equal to the value of the right operand, then the condition is **true.**	(X >= Y) is **false.**
<=	If the value of the left operand is less than or equal to the value of the right operand, then the condition is **true.**	(X <= Y) is **true.**

For example:

```
x = 9
y = 13
print('x > y is',x > y) # Here 9 is not greater than 13
print('x < y is',x < y) # Here 9 is less than 13
print('x == y is',x == y) # Here 9 is not equal to 13
print('x != y is',x != y) # Here 9 is not equal to 13
print('x >= y is',x >= y) # Here 9 is not greater than or equal to 13
print('x <= y is',x <= y) # Here 9 is less than 13
```

Output :

```
x > y is False
x < y is True
x == y is False
```

```
x != y is True
x >= y is False
x <= y is True
```

Logical Operators

The operators that act on one or two boolean values and return another boolean value are called logical operators. There are 3 key logical operators. Let X and Y be the two operands and **X = True** and **Y = False.**

Operator	Description	Example
and	Logical AND: If both the operands are true then the condition is true.	(X and Y) is **false**
or	Logical OR: If any of the two operands are then the condition is true.	(X or Y) is **true**
not	Logical NOT: Used to reverse the logical state of its operand.	Not(X) is **false**

The Truth table for all combinations of values of X and Y

X	Y	X and Y	X or Y	not(X)	not(Y)

T	T	T	T	F		F
T	F	F	T	F		T
F	T	F	T	T		F
F	F	F	F	T		T

Introduction to If-Else

There are certain points in our code when we need to make some decisions and then based on the outcome of those decisions we execute the next block of code. Such conditional statements in programming languages control the flow of program execution.

The most commonly used conditional statements in Python are

- Simple If statements
- If-Else statements
- If-Elif statements
- Nested Conditionals

Simple If statements

These are the most simple decision-making/conditional statements. It is used to decide whether a certain statement or block of statements will be executed or not.

- The most important part of any conditional statement is a condition or a boolean.
- And the second important thing is the code block to be executed.

In the case of simple `If` statements, if the conditional/boolean is true then the given code block is executed, else the code block is simply skipped and the flow of operation comes out of this If condition.

The **general syntax** of such statements in Python is:

```
if <Boolean/Condition>:

     <Code Block to be executed in case the Boolean is True>

<Code Block to be executed in case the Boolean is False>
```

An **example** of a simple **if** statement can be as follows:

```
Val = False
if Val == True:
     print("Value is True") # Statement 1
print("Value is False") # Statement 2
```

In the above code, the variable Val has a boolean value **False**, and hence the condition is not satisfied. Since the condition is not satisfied, it skips the If statement, and instead, the next statement is executed. Thus the output of the above code is:

```
Value is False
```

Importance of Indentation in Python

To indicate a block of code and separate it from other blocks in Python, you must indent each line of the block by the same amount. The two statements in our example of Simple if-statements are both indented four spaces, which is a typical amount of indentation used in the case of Python.

In most other programming languages, indentation is used only to improve the readability of the code. But in Python, it is required to indicate what block of code, a statement belongs to.

For instance, *Statement 1* which is indented by 4 spaces is a part of the **if** statement block. On the other hand, *Statement 2* is not indented, and hence it is not a part of the **if** block. This way, indentation indicates which statements from the code belong together.

Any deviation from the ideal level of indentation for any statement would produce an indentation error. **For example:** On running the given script:

```
Val = False
if Val == True:
```

We get the output as

```
IndentationError: unindent does not match any outer
indentation level
```

This error is because *statement 1* is not in the indentation line for the **if** statement.

Else-If statements

The simple **if** statement, tells us that if a condition is true it will execute a block of statements, and if the condition is false it won't. But what if we want some other block of code to be executed if the condition is false? Here comes the *else* statement. We can use the **else** statement with the **if** statement to execute a block of code when the condition is false. The general Syntax for the If-Else statement is:

```
if (Condition/Boolean):
    <Code block to be executed in case the condition is True>
else:
    <Code block to be executed in case the condition is
```

```
False>
```

Keep in mind the indentation levels for various code blocks.

Let us now take an example to understand If-Else statements in depth.

Distinguishing between Odd and Even numbers:

Problem Statement: Given a number, print whether it is odd or even.

Approach: For a number to be even, it must be divisible by 2. This means that the remainder upon dividing the number by 2 must be 0. Thus, to distinguish between odd and even numbers, we can use this condition. The numbers that leave a remainder 0 on division with 2 will be categorized as even, else the number is odd. This can be written in Python as follows:-

```
num = 23
If num%2 == 0:
    print("Even Number")
else:
    print("Odd Number")
```

The output of this code will be:-

```
Odd Number
```

Since 23 is an odd number, it doesn't satisfy the **if** condition, and hence it goes to **else** and executes the command.

If-Elif-Else statements

So far we have looked at Simple If and a single If-Else statement. However, imagine a situation in which if a condition is satisfied, we want a particular block of code to be executed, and if some other condition is fulfilled we want some other block of code to run. However, if none of the conditions is fulfilled, we want some third block of code to be executed. In this case, we use an **if-elif-else** ladder.

In this, the program decides among multiple conditionals. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

The general syntax of an **if-elif-else** ladder will be:

```
if <Condition 1>:
     <Execute this code block>
elif <Condition 2>:
     <Execute this code block>

elif <Condition X>:
     <Execute this code block>
else:
     <Execute this code block>
```

Keep in mind the indentation levels for various code blocks.

Note:- We can have as many elif statements as we want, between the if and the else statements. This means we can consider as many conditions as we want. It should be noted that once an if or elif condition is executed, the remaining elif and else statements will not be executed.

Finding the largest among three numbers

Problem Statement: Given three numbers A, B, and C, find the largest among the three and print it.

Approach: Let A, B, and C, be 3 numbers. We can construct an **if-elif-else** ladder. We have to consider the following conditions:

- If A is greater than or equal to both B and C, then A is the largest Number.
- If B is greater than or equal to both A and C, then B is the largest number.
- However, if none of these conditions is true it means that C is the largest number.

Thus, we get the following implementation for the above approach.

```
A = 10
B = 20
C = 30
if A>=B and A>= C:
     print(A)
elif B>=C and B>=A:
     print(B)
else:
     Print(C)
```

- Here since 10 is not greater than 20 and 30, the first **if** the condition is not satisfied. The code goes on to the **elif** condition.
- Now, 20 is also not greater than both 10 and 30, thus even the **elif** condition is not true. Thus, the else code block will now be executed.
- Thus the output will be 30, as it is the largest among the three. The **else** conditional block is executed.

The output of the above code will be :

```
30
```

Nested Conditionals

A nested **if** is an **if** statement that is present in the code block of another **if** statement. In other words, it means- an **if** statement inside another **if** statement. Yes, Python allows such a framework for us to nest **if** statements. Just like nested **if** statements, we can have all types of nested conditionals. A nested conditional will be executed only when the parent conditional is true.

The general syntax for a very basic nested if statement will be:

```
if <Condition 1>:
    # If Condition 1 is true then execute this code block
    if <Condition 2>:
        < If Condition 2 is True then execute this code
block>
    else:
       < If Condition 2 is False then execute this code
block>

else:
    # If Condition 1 is False then execute this code block>
    if <Condition 3>:
       < If Condition 3 is True then execute this code block>
    else:
       < If Condition 3 is False then execute this code
block>
```

Note:- The conditions used in all of these conditional statements can be comprised of relational or logical operators. For example:-

```
A = True
B = False
if( A and B ):   # True and False = False
     print("Hello")
else:
     print("Hi") # This code block will be executed
```

The output of the above code will be:

```
Hi
```