Logical Operators

Logical operators are used to construct more complex expressions. You can make complex comparisons by joining comparison statements together using the logical operators: **and**, **or**, **not**. Complex comparisons return a Boolean (**True** or **False**) result.

- **and**
  - Both sides of the statement being evaluated must be True for the whole statement to be True.
  - Example: (5 > 1 **and** 5 **<** 10) = **True**
- **or**
  - If either side of the comparison is True, then the whole statement is True.
  - Example: (color = "blue" **or** color = "green") = **True**
- **not**
  - Inverts the Boolean result of the statement immediately following it. So, if a statement evaluates to True, and we put the not operator in front of it, it would become False.
  - Example: (**not** "A" **==** "A") = **False**

PART 1: The **and** Logical Operator

In Python, you can use the logical operator **and** to connect more than one comparison. This type of complex comparison is used to check if two comparison statements are both True or not. You might use the **and** operator when you need to execute a block of code, but only if two different conditions are true. For example, you might want to write a script that automates sending you an emergency alert if a server stops responding *and* there is an unusual increase in employees opening trouble tickets.

Example 1:

The following model demonstrates the use of the **and** logical operator to join comparisons between two mathematical expressions. The description below the example explains the order in which Python will process the line of code.

```
# Example 1

print((6*3 >= 18) and (9+9 <= 36/2))
```

Run

Reset

In the example above, the following activities were completed by Python in the following order:

1. Python solves the numerical expressions using the order of operations. **(6*3 >= 18) and (9+9 <= 36/2) becomes (18 >= 18) and (18 <= 18)**
2. Python compares the results of the numerical expressions using the comparison operators (in this case >= and <=). **(18 >= 18) and (18 <= 18) becomes True and True**
3. Python checks if both sides of the logical operator "and" are true. **True and True become True**
4. Python returns a Boolean value: True or False. **The complex comparison returns a True result.**

Example 2:

In this next example, "Nairobi" < "Milan" and "Nairobi" > "Hanoi", the **and** logical operator is connecting two string comparison statements. You learned previously that using the greater than and less than operators on strings will test the alphabetical order (technically Unicode values) of the strings. So, this complex comparison is checking if "Nairobi" is alphabetized before "Milan" (False) AND after "Hanoi" (True).

This comparison returns a False result because both sides of the logical operator are not True. A comparison statement like this might be used to iterate through a list of names to check if they are alphabetized in the correct order.

```
# Example 2

print("Nairobi" < "Milan" and "Nairobi" > "Hanoi")
```

Run

Reset

PART 2: The **or** Logical Operator

The **or** logical operator tests two conditions to determine if at least one side of the **or** logical operator is True. The result of the test can be used to trigger a block of code if at least one condition is present.

Syntax:

```
Expression1 or Expression2
```

Returns Booleans:

Expression1	Expression2	Returns Result
True	True	True
True	False	True
False	True	True
False	False	False

Examples:

```
# True or True returns True
print((15/3 < 2+4) or (0 >= 6-7))
True

# False or True returns True
print(country == "New York City" or city == "New York City")
True

# True or False returns True
print(16 <= 4**2 or 9**(0.5) != 3)
True

# False or False returns False
print("B_name" > "C_name" or "B_name" < "A_name")
False
```

PART 3: The **not** Logical Operator

The **not** logical operator inverts the value of the comparison expression. This is a helpful tool when you want to execute a block of code as long as a certain condition is **not** present.

- If the conditional expression is True, the **not** logical operator can be added to make the expression **not** True (False).
- If the conditional expression is False, the **not** logical operator can be added to make the expression **not** False (True).

Syntax:

```
not expression
```

Example 1:

```
# Test Example 1:

x = 2*3 > 6
print("The value of x is:")
print(x)

print("")  # Prints a blank line

print("The inverse value of x is:")
print(not x)
```

Run

Reset

Example 2:

Can you determine the result of the following comparison?

```
# What happens when you negate a False statement?
# Click Run when you are ready to check your answer.


today = "Monday"
print(not today == "Tuesday")


# The "today" variable states today is Monday. This makes the comparison
# "today == Tuesday" False. The logical operator "not" inverts the False
# result to become True. In other words, this expression asks if it is
# false that today is not Tuesday. More succinctly, "not False" means
# True."

```

Run

Reset

PART 4: Practice

If you would like more practice using the logical (**and**, **or**, **not**) operators, feel free to create your own comparisons using the code block below. Note that there is no feedback associated with this code block.

```

```

Run

Reset

For additional Python practice, the following links are for several popular online interpreters and codepads:

- Welcome to Python