Python Programming

Logical Operators

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+x, -x, ~x	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	==, !=, <, <=, >, >=, is, is not	Comparison, identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

Logical operators

```
# You can use () to force priority
age, salary = 30, 7000

result = (age > 25) and (salary < 8000) # True

# but > is higher in precedence than and, so parentheses can be removed

result = age > 25 and salary < 8000 # True
print(result)

print(age > 25 and salary > 9000)   # False

print(age > 35 or salary < 8000)   # True

print(age > 35 or salary > 9000)   # False
```

Mixing Logical Operators

```
2   # priority: not, and, or
3   age, salary, weight = 30, 7000, 110
4
5   print(age > 25 and salary < 8000 and weight < 150)  # True
6   print(age > 25 and salary < 8000 and weight > 200)  # False
7   print(age > 35 or salary > 6000 or weight > 200)    # True
8   print(age > 35 and salary > 6000 or weight > 200)   # False
9   print(age > 20 and salary > 6000 or weight > 200)   # True
10
11  status = weight >= 150
12  print(age > 25 and salary < 8000 and not status)    # True
13
14
```

() applied first

```
3   age, salary, weight = 30, 7000, 110
4
5   print(age > 25 or salary > 6000 and weight > 200)    # True
6
7   # change evalaution order with ()
8   print((age > 25 or salary > 6000) and weight > 200) # False
```

Let’s try simplifying

- Let’s simplify this expression T and T and (F or (T and T)) or T
- T and T and (F or **(T and T)**) or T ⇒ **(T and T)** is the simplest (). Its value is T
- T and T and (**F or T**) or T ⇒ (**F or T**) is the simplest (). Its value is T
- T and T and T or T ⇒ No more (). Next is group ands
- **T and T and T** or T ⇒ **T and T and T** is group of ands. Evaluate to T
- T or T. Now final expression is set of conditions ORed ⇒ T

Chaining comparison operators

- A very nice way to simplify expressions in a logical way
- Similar to assignment operator: Non associative operators
  - Just implemented as series of comparisons

```
3   a, b, c, d = 10, 20, 30, 40
4
5   # a < b < c < d
6   # same as
7   # a < b and b < c and c < d
8   print(a < b < c < d)    # True
9   print(a < b < d > c)    # True
10  print(a < b < d < c)    # False
11
12  print(a == b == c == d) # False
13
```

Short-Circuit Evaluation

- Stop evaluating when result is determined (**efficiency**)
- Let say we have an expression
- T or T and (F or (T and T)) or T ⇒ (T and T)
  - Do we really need to evaluate after the first T or <something>
  - No. According to OR table, this is **definitely** TRUE. STOP
- T and T and F and (F or (T and T) and (F or (T and T)))
  - Do we really need to evaluate after the first T and T and F and <something>
  - No. According to AND table, this is **definitely** False. STOP
- Note: In complex expression: some sub-groups are discarded, but still continue evaluating
  - Rule: Logically discard what can be discarded, following the precedence rules

Short-Circuit Evaluation

```
2   # both expressions evaluated
3   print(2 > 10 or 5 < 6)      # True
4
5   # Expression 5 < 6 will be discarded
6   print(2 < 10 or 5 < 6)      # True
7
8   # Expression 3 < 6 will be discarded
9   print(2 > 10 and 3 < 6 or 3 > 4)    # False
10
11
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."