**Short-circuiting** in Python and many other programming languages optimizes the evaluation of logical expressions.

condition is terminated as soon as the outcome is determined, avoiding unnecessary computations.

It enhances program efficiency, especially in **if-else** statements, **switch** cases, and **loops** with conditional operations.

This mechanism primarily applies to logical **and** and **or** operations in Python.

Short-circuiting helps prevent issues like division by zero and operations on null objects, making the code more robust.

**AND opearation**: In the case of the **AND** operator, if the first operand is False, the overall result is False, and Python skips evaluating the second operand.

Example 1:

|  |
| --- |
| x = 0  y = 5  if x != 0 **and** y%x == 0:  print(" y is divisible x")  else:  print(" y is not divisible x OR division is not possible ") |

**Exaple 2:**

|  |
| --- |
| x = None      y = " World !"      if x is not None **and** y is not None:          result = x + y          print(result) |

Here, the division by zero is avoided because the first condition (x != 0) is False, so the second condition isn't evaluated and avoid happening ‘division by zero’ or string concatenation operation on ‘NoneType’ variable value **x** errors.

**OR operation**: For the **OR** operator, if the first operand is True, the overall result is True, and the second operand is ignored. For example:

|  |
| --- |
| i =0      j= 10      while i < j or j >= i:          print(f"between i and j: i={i} , j={j}")          i +=1          j -=1 |

In this scenario, the condition **i < j** remains **True** until **i = 5 and j = 5**. Because i is always less than j before this point, the second part of the or condition is never evaluated. However, when **i = 5 and j = 5**, the first condition becomes **False**, prompting the interpreter to check the second condition (**j >= i**). Since this evaluates to **True**, the overall condition is satisfied and becomes **True**.

Short-circuiting enhances performance by skipping unnecessary evaluations and helps prevent runtime errors, such as division by zero or accessing undefined variables.