Python and Operator

Summary: in this tutorial, you'll learn about the Python `and` logical operator and how to use it to control the flow of code.

Introduction to the Python and operator

The Python `and` operator is a logical operator (https://www.pythontutorial.net/python-basics/python-logical-operators/) . Typically, you use the `and` operator to operate on Boolean values and return a Boolean value (https://www.pythontutorial.net/python-basics/python-boolean/) .

The `and` operator returns `True` if both operands evaluate to `True` . Otherwise, it returns `False` .

The following truth table shows the result of the `and` operator:

X	Y	X and Y
True	True	True
True	False	False
False	False	False

X	Y	X and Y
False	True	False

This table illustrates two important points:

- If the first operand (`X`) is `True` , the result of the `and` operator depends on the result of the second operand (`Y`).
- If the first operand (`X`) is `False` , the result of the `and` operator is always `False` regardless of the value of the second operand (`Y`).

The following shows an example of using the `and` operator:

```
timeout = False
pending_job = True
execute_next = timeout and pending_job

print(execute_next)
```

Output:

```
False
```

In this example, the `timeout` is `False` and `pending_job` is `True` . Therefore, the result of the expression `timeout and pending_job` is `False` .

Python and operator is short-circuiting

The key feature of the `and` operator is that it short-circuits. It means that if the first operand is `False` , the `and` operator won't evaluate the second operand. The reason is that it already has a conclusion about the outcome, which is `False` .

The following example results in a `ZeroDivisionError` :

```
a = 10
b = 0
c = a / b
print(c)
```

In this example, since `b` is zero, the `a / b` definitely causes the division by zero exception.

However, the following example won't cause a `ZeroDivisionError`:

```
a = 10
b = 0
c = b and a/b
print(c)
```

Output:

```
0
```

In this example, we used the `and` operator in the expression:

```
c = b and a/b
```

Since `b` is zero, which is `False`, the `and` operator can conclude the result of the whole expression, which is `False` regardless of the result of the second part `a / b`. Therefore, the `and` operator doesn't need to evaluate the expression `a / b`. In fact, it doesn't do so.

The following example changes the value of `b` to five:

```
a = 10
b = 5
c = b and a/b
print(c)
```

Output:

```
2.0
```

In this example, `b` is `5` which is `True`. Since the first operand is `True`, the value of the whole expression depends on the value of the second operand, which is `a / b`.

These examples show that the `and` operator can operate with non-Boolean values and returns a non-value boolean value.

In general, you can use the `and` operator for the objects:

```
bool(object1) and bool(object2)
```

In fact, you don't need to use the `bool()` (https://www.pythontutorial.net/advanced-python/python-bool/) constructor:

```
object1 and object2
```

In this case, the `and` opeartor returns the `object1` if it's falsy. Otherwise, it returns the `object2`.

In other words, the `and` operator returns the `object1` if the `object1` is falsy. Otherwise, it evaluates the `object2` and returns it.

The expression

```
c = b and a/b
```

is equivalent to the following:

```
if b:
    c = a / b
else:
    c = b
```

By using the `and` operator, you can control the flow of the program.

Python and operator example

The following defines the `avg()` function that calculates the average of numbers:

```
def avg(*numbers):
    total = sum(numbers)
    n = len(numbers)
    if n > 0:
        return total / n
    return 0


if __name__ == "__main__":
    print(avg(1, 2, 3))
```

Output:

```
0
```

How it works.

- First, calculate the sum of the numbers using the `sum()` function.
- Second, get the number numbers using the `len()` function.
- Third, return the average if the number of numbers is greater than zero, otherwise, return zero.

The main block calculates the average of three numbers 1, 2, and 3, which returns 2.0 as expected.

The following code uses an `if` statement (https://www.pythontutorial.net/python-basics/python-if/) and returns the average if the number of numbers is greater than zero. Otherwise, it returns zero:

```
if n > 0:
    return total / n
```

```
    return 0
```

In fact, you can use the and operator to make this code more concise like this:

```
    return n and total / n
```

In this case, if `n` is zero the and operator doesn't need to evaluate the expression `total / n` and it returns zero. Otherwise, it evaluates the `total / n` and returns it.

Here is the new version of the `avg()` function that uses the `and` operator:

```
def avg(*numbers):
    total = sum(numbers)
    n = len(numbers)

    return n and total / n
```

Summary

- The `X and Y` returns `True` if both `X` and `Y` evaluate to `True`. Otherwise, it returns `False`.
- The `X and Y` actually returns `X` if `X` is falsy. Otherwise, it evaluates `Y` and returns the result of the evaluation.