Секреты логических операторов в Python

Средний 8 мин 21K

Python*, Программирование*

Логические операции играют важную роль в программировании. Они используются для создания условных конструкций и составления сложных алгоритмов. В Python для выполнения логических операций используются **логические операторы:**

- `not` — логическое отрицание
- `and` — логическое умножение
- `or` — логическое сложение

Таблицы истинности логических операторов

Мы привыкли к тому, что обычно в языках программирования логические операторы возвращают значения `True` или `False` согласно своим таблицам истинности.

Таблица истинности оператора `not`:

a	not a
`False`	`True`
`True`	`False`

Таблица истинности оператора `and`:

a	b	a and b
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

Таблица истинности оператора `or` :

a	b	a or b
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`True`

Когда операндами логических операторов являются объекты `True` и `False`, работа логических операторов в Python также соответствует данным таблицам истинности.

Приведенный ниже код:

```
print(not True)
print(not False)
print(False and True)
print(True and True)
print(False or True)
print(False or False)
```

Чтобы привести объекты к значению `True` или `False`, используется встроенная функция `bool()`.

Приведенный ниже код:

```
# falsy объекты
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(0.0))
print(bool([]))
print(bool(''))
print(bool({}))

#truthy объекты
print(bool(True))
print(bool(123))
print(bool(69.96))
print(bool('beegeek'))
print(bool([4, 8, 15, 16, 23, 42]))
print(bool({1, 2, 3}))
```

Оператор not

Как мы уже знаем, операндом оператора `not` может быть объект любого типа. Если операнд отличен от значений `True` и `False`, он оценивается в соответствии с концепцией `truthy` и `falsy` объектов. При этом результатом работы оператора `not` **всегда является** значение `True` или `False`.

Приведенный ниже код:

```
print(not False)
print(not None)
print(not 0)
print(not 0.0)
print(not [])
print(not '')
print(not {})
```

Операторы and и or

Операндами операторов `and` и `or`, как и в случае с `not`, могут быть объекты любых типов данных. По аналогии с оператором `not` можно предположить, что результатом работы логических операторов `and` и `or` также является значение `True` или `False`. Однако на самом деле данные операторы возвращают **один из своих операндов**. Какой именно — зависит от самого оператора.

Приведенный ниже код:

```
print(None or 0)
print(0 or 5)
print('beegeek' or None)
print([1, 2, 3] or [6, 9])

print(1 or 'beegeek' or None)
print(0.0 or 'habr' or {'one': 1})
print(0 or '' or [6, 9])
print(0 or '' or [])
print(0 or '' or [] or {})
```

Аналогично дело обстоит с оператором `and` .

Приведенный ниже код:

```
print(None and 10)
print(5 and 0.0)
print('beegeek' and {})
print([1, 2, 3] and [6, 9])

print(1 and 'beegeek' and None)
print('habr' and 0 and {'one': 1})
print(10 and [6, 9] and [])
```

Приоритет логических операторов

Важно помнить о приоритете логических операторов. Ниже логические операторы представлены в порядке уменьшения приоритета (сверху вниз):

1. `not`
2. `and`
3. `or`

Согласно приоритету логических операторов приведенный ниже код:

```
a = 0
b = 7
c = 10
print(not a and b or not c)        # 7
```

эквивалентен следующему:

```
a = 0
b = 7
c = 10
print(((not a) and b) or (not c))  # 7
```

По отношению к другим операторам Python (за исключением оператора присваивания `=`) **логические операторы имеют самый низкий приоритет**.

Выполняйте домашнюю работу, вам понравится!

А теперь … успехов!