В языках программирования логический тип - это обычный способ представления чего-либо, имеющего только два противоположных состояния, например, *включено* или *выключено*, *да* или *нет* и т.д. Это очень полезный тип, в котором вы быстро убедитесь сами, когда начнете работать над своими проектами или даже писать небольшие программы. В этом разделе мы рассмотрим логический тип в Python и узнаем, как его использовать.

**Логический тип**

**Логический** тип, или просто bool, представляет собой особый тип данных, который имеет только два возможных значения: True и False. В Python имена логических значений начинаются с заглавной буквы.

Если вы пишете приложение, которое отслеживает открывание дверей, вы сочтете естественным использовать bool для хранения текущего состояния двери.

is\_open = True

is\_closed = False

print(is\_open) # True

print(is\_closed) # False

**Булевы операции**

В Python есть три встроенных логических оператора: and, or и not. Первые два являются **бинарными** операторами, что означает, что они ожидают два аргумента. not является **унарным** оператором, он всегда применяется к одному операнду. Сначала давайте посмотрим на эти операторы, применяемые к логическим значениям.

* **и** является двоичным оператором, он принимает два аргумента и возвращает True, если оба аргумента истинны, в противном случае он возвращает False.
* a = True and True # True
* b = True and False # False
* c = False and False # False

d = False and True # False

* **or** является двоичным оператором, он возвращает, True если хотя бы один аргумент имеет значение true, в противном случае он возвращает False.
* a = True or True # True
* b = True or False # True
* c = False or False # False

d = False or True # True

* **not** является унарным оператором, он изменяет логическое значение своего аргумента на обратное.
* to\_be = True # to\_be is True

not\_to\_be = not to\_be # not\_to\_be is False

**Приоритет логических операций**

Логические операторы имеют другой приоритет, и это влияет на порядок вычисления. Вот операторы в порядке их приоритетов:

1. not

2. and

3. or

Итак, not сначала рассматривается, затем and, наконец or. Имея это в виду, рассмотрим следующее выражение:

print(False or not False) # True

Сначала оценивается часть not False, и после оценки мы остаемся с False or True. Это приводит к True, если вы помните предыдущий раздел.

Хотя работа исключительно с логическими значениями может показаться очевидной, приоритет логических операций будет очень важно помнить, когда вы начнете работать с так называемыми **истинными** и **ложными** значениями.

**Истинные и ложные значения**

Хотя Python имеет логический тип данных, мы часто хотим использовать не булевы значения в логическом контексте. А Python позволяет проверять практически любой объект на достоверность. При использовании с логическими операторами значения не булевых типов, такие как целые числа или строки, называются **истинными** или **ложными**. Это зависит от того, интерпретируются ли они как True или False.

Следующие значения оцениваются как False в Python:

* константы, определенные как false: None и False,
* ноль любого числового типа: 0, 0.0,
* пустые последовательности и контейнеры: "", [], {}.

Все остальное обычно оценивается как True. Вот пара примеров:

print(0.0 or False) # False

print("True" and True) # True

print("" or False) # False

Вообще говоря, and и or могут принимать любые аргументы, которые могут быть проверены на логическое значение.

Теперь мы можем более наглядно продемонстрировать разницу в приоритете операторов:

# `and` has a higher priority than `or`

truthy\_integer = False or 5 and 100 # 100

Опять же, давайте разберем приведенное выше выражение на части. Поскольку оператор and имеет более высокий приоритет, чем or, мы должны рассмотреть 5 and 100 часть. Оба 100 и 5 являются истинными значениями, поэтому эта операция вернет 100. Вы никогда не видели этого раньше, поэтому естественно задаться вопросом, почему у нас здесь число вместо True значения. Объяснение заключается в следующем:

Операторы or и and возвращают один из своих операндов, не обязательно логического типа (см. Подробности ниже). Тем не менее, not всегда возвращает логическое значение.

* and возвращает первое значение, если оно равно False, в противном случае возвращает второе значение.

>>> False and True

False

>>> True and True

True

>>> True and False

False

* or возвращает первое значение, если оно равно True, в противном случае возвращает второе значение.

>>> True or False

True

>>> False or True

True

>>> True or True

True

>>> False or False

False

Возвращаясь к исходному выражению, вы можете видеть, что последняя часть False or 100 делает точно то же самое, возвращает 100 вместо True.

Ниже приведен еще один сложный пример:

tricky = not (False or '') # True

Пара круглых скобок - это способ указать порядок, в котором выполняются операции. Таким образом, сначала мы оцениваем эту часть выражения: False or ''. Этот операнд '' вычисляется как False и or возвращает эту пустую строку. Поскольку результат вложенного выражения отрицается, мы получаем True в конце: not '' то же самое, что True. Почему мы не получили, скажем, непустую строку? not Оператор создает новое значение, которое по умолчанию имеет логический тип. Итак, как указывалось ранее, унарный оператор всегда возвращает логическое значение.

**Оценка короткого замыкания**

Последнее, что следует упомянуть, это то, что логические операторы в Python имеют **короткое замыкание**. Вот почему их также называют **ленивыми**. Это означает, что второй операнд в таком выражении вычисляется только в том случае, если первого недостаточно для вычисления всего выражения.

* x and y возвращает x, если значение x неверно; в противном случае он оценивает и возвращает y .
* x or y возвращает x, если x соответствует действительности; в противном случае он оценивает и возвращает y .

Например:

# division is never evaluated, because the first argument is True

lazy\_or = True or (1 / 0) # True

# division is never evaluated, because the first argument is False

lazy\_and = False and (1 / 0) # False

**Краткие сведения**

Итак, в этом разделе мы узнали о логическом типе в Python, о том, какие логические операции он выполняет (not, and, or) и их приоритете. Мы также ознакомились с концепцией истинных и ложных значений и почему логические операторы в Python называются короткозамкнутыми. Это были самые основы булевых значений и логических операций в Python. Определенно полезно знать их с самого начала!