Python: Результат логических выражений

- Правила преобразования
- Составные выражения
- Двойное отрицание
- Ошибка выбора

В этом уроке познакомимся с правилами преобразования аргумента и узнаем, как работать с составными выражениями и двойным отрицанием.

Правила преобразования

Посмотрите на пример:

```
print(0 or 1)
```

```
1
```

Оператор **ИЛИ** работает так, что его выполнение слева направо прерывается и возвращается результат первого аргумента, который можно преобразовать в `True`. Если такого аргумента нет, возвращается последний — правый.

Пример с оператором **И**:

```
print(0 and 1)
```

```
0
```

Оператор **И** работает так, что его выполнение слева направо прерывается и возвращается результат первого аргумента, который можно преобразовать в `False`. Если такого аргумента нет, возвращается последний — правый.

В Python есть два правила преобразования:

Этими правилами пользуются в разработке, например, чтобы определить значение по умолчанию:

```
value = name or ''
# Примеры
234 or '' # 234
'hexlet' or '' # 'hexlet'
None or '' # ''
```

Если `name` примет одно из falsy-значений, переменной `value` будет присвоена пустая строка. В этом случае в последующем коде мы сможем работать с `value` как со строкой.

Но здесь есть потенциальный баг. Если `name` содержит falsy значение, а переменной `value` можно присвоить значения типа `0`, `False`, `None`, то код выше заработает

неверно:

```
# Значение на самом деле есть,
# но оно Falsy, поэтому не выбирается на условии OR
False or '' # ''
0 or '' # ''
None or '' # ''
```

Составные выражения

Если соединить логические выражения между собой, можно получать довольно интересные способы решения задач с кодом.

Допустим, нам нужно реализовать код, в котором в переменную записывается:

Это можно сделать, если использовать знания, полученные выше:

```
# число четное
result = 10 % 2 == 0 and 'yes' or 'no' # 'yes'
# или сразу печатаем на экран
print(10 % 2 == 0 and 'yes' or 'no') # => 'yes'
# число нечетное
print(11 % 2 == 0 and 'yes' or 'no') # => 'no'
```

Эти выражения работают согласно порядку и приоритетам. Приоритет присваивания самый низкий, поэтому оно происходит в конце. Приоритет

сравнения == выше, чем приоритет логических операторов `and` и `or`, поэтому сравнение происходит раньше. Дальше код выполняется слева направо, так как приоритет `and` выше, чем приоритет `or`. Рассмотрим по шагам:

```
# Для четного
# 1 шаг
10 % 2 == 0 # True
# 2 шаг
True and 'yes' # Результат — истина
# Проверка на or выполняется, но правая часть не исполняетс

# Для нечетного
# 1 шаг
11 % 2 == 0 # False
# 2 шаг
False and 'yes' # Результат — ложь, проверяем дальше
# 3 шаг
False or 'no' # Выбирается и возвращается 'no'
```

Такую же схему можно использовать с любым выражением в начале:

```
print(somefunc() and 'yes' or 'no')
```

Можете проверить себя и поэкспериментировать с кодом в Replit.

Двойное отрицание

Напомним, как выглядит операция отрицания:

```
answer = True
print(not answer)  # => False
```

При двойном отрицании итоговое значение равно начальному:

```
answer = True
print(not not answer)  # => True
```

Оператор `not` всегда возвращает булевое значение, независимо от типа переданного аргумента, а не заменяет значение на противоположное. Поэтому двойное отрицание тоже вернет булевое True/False.

```
answer = 'python'
print(not answer) # => False
print(not not answer) # => True
```

Ошибка выбора

Представьте, что нам нужно проверить — значение равно одному или другому. Например, переменная `value` должна содержать одно из двух значений: `first` или `second`. Начинающие разработчики иногда записывают это выражение так:

```
value == ('first' or 'second')
```

Однако такой код приведет к неверному результату. Необходимо вспомнить приоритет выполнения операций. Первым делом вычисляется все, что указано в скобках — `'first' or 'second'`. Если выполнить этот код в Replit, то вывод будет таким:

```
python
Python 3.8.2 (default, Apr 12 2020, 15:53:37)
>>> 'first' or 'second'
'first'
>>>
```

Теперь заменим исходное выражение на частично вычисленное:

```
value == 'first'
```

Совсем не то, что мы ожидали. А теперь вернемся к началу и напишем проверку правильно:

```
# Скобки ставить не обязательно,
# потому что приоритет == выше чем приоритет or
value == 'first' or value == 'second'
```

Задание

Реализуйте функцию `string_or_not()`, которая проверяет является ли переданный параметр строкой. Если да, то возвращается `'yes'` иначе `'no'`

```
string_or_not('Hexlet') # 'yes'
string_or_not(10) # 'no'
string_or_not('') # 'yes'
string_or_not(False) # 'no'
```

Проверить то, является ли переданный параметр строкой, можно при помощи функции isinstance():

```
isinstance(3, str) # False
isinstance('Hexlet', str) # True
```

Поэкспериментируйте с кодом в интерактивном репле https://replit.com/@hexlet/python-basics-logical-expressions

- ▶ Упражнение не проходит проверку — что делать? 😶
- ▶ В моей среде код работает, а здесь нет 😐
- ▶ Мой код отличается от решения учителя 🤔
- ▶ Прочитал урок — ничего не понятно 🙄

Полезное

- Boolean