ЛАБОРАТОРНАЯ РАБОТА

Тема: ***Литералы. Операторы сравнения и логические выражения***

Цель работы: Знакомство с типами литералов, используемых в программах на Питоне, логическим типом, операторами сравнения и правилами построения и вычисления логических выражений.

Содержание

Числовые литералы 1
Строковые литералы 2
Экранирование и escape-последовательности 3
ЗАДАНИЕ 1 3
Простейшие операции со строковыми литералами 4
ЗАДАНИЕ 2 5
Логический тип данных 6
Операторы сравнения 6
ЗАДАНИЕ 3 7
Логические выражения 7
Числа и строки как операнды логических выражений 8
Сокращенные вычисления логических выражений 8
Логические операторы и условные выражения 9
ЗАДАНИЕ 4 9
Вопросы для самоконтроля 10
Escape-последовательности 10
Старшинство операторов 11

Программа — это *текст*, который строится из

- служебных слов,
- имен различных объектов,
- знаков операций (операторов),
- разделителей (скобки, запятые и т.п.) и
- обозначений констант.

Каждая константа представляется несколькими символами текста, которые образуют *литерал* (англ. *literal* — буквенная константа): `123`, `"Питон"`, `True`.

Анализируя программу, интерпретатор выделяет литералы, распознает их типы по виду и преобразует во внутреннее представление

В программах на Питоне используется три основных типа литералов, представляющих *числовые значения*, *строки* и *логические константы*.

Числовые литералы

Для чисел выделяются три типа литералов: *целые*, *нецелые* (с десятичной точкой) и *мнимые*.

Если в записи *литерала из цифр* встречается десятичная точка, то интерпретатор трактует литерал как запись вещественного значения, *даже если дробная часть равна нулю или отсутствует* (когда нуль в дробной части подразумевается). Например,

`1` интерпретируется как целое число,

`1.` `1.0` `0.1` и `.1` — как вещественные.

Литералы целых чисел могут представлять числа в разных системах счисления: десятичной, двоичной, восьмеричной и шестнадцатеричной.

Мнимые числа распознаются по наличию в конце числового литерала символа `j` или `J`.

Знак перед числом в тексте программы *не считается* частью литерала. Например, запись `-1` — это выражение, составленное из символа операции смены знака `'-'` и следующего за ним литерала `1`. Это учитывается при разборе текста программы, и для

обычных чисел такая трактовка особой роли не играет. Но, из этого свойства вытекает, что в Питоне нет литералов комплексных чисел. Они конструируются из двух литералов (числового и мнимого типов), соединенных знаком плюс или минус.

```
>>> 2j
2j        # литерал типа "мнимое число", выводится без скобок
>>> 1+2j
(1+2j)    # комплексное число, образовано из двух литералов
```

Строковые литералы

Символ — термин, который применяется для обозначения отдельной буквы, цифры или знака пунктуации.

Из групп символов формируются *символьные строки*. Символьные строки, предназначенные для отображения текста, называются строковыми литералами или просто строками.

Встретив в инструкции строковый литерал, для последующей обработки Питон преобразует его в специальный строковый объект данных (в Python 3.x типа `str`). По умолчанию применяется кодировка UTF-8.

Если фрагмент текста предназначен для буквального отображения, то его нужно заключить в кавычки. Для этого в Питоне предлагаются две разные синтаксические формы.

а) Однострочные литералы.

Чтобы образовать литерал из одной строки текста (однострочный литерал), достаточно заключить текст в одинарные или двойные кавычки. Эти кавычки *не считаются частью литерала*, так как играют роль ограничителей литерала.

Два равноценных варианта синтаксиса оказываются полезны, когда текст литерала содержит кавычки. В строке, заключенной в двойные кавычки, могут встретиться одиночные кавычки, и наоборот. Например,

`"Система 'Windows' установлена"` или `'Система "Windows" установлена'`.

В случае вложенных кавычек необходимо соблюдать порядок открытия и закрытия пар кавычек.

Литералы, не содержащие ни одного символа (даже пробела), называются *пустыми строками* (`''` или `""`).

Когда строка является результатом вычисления выражения и ней нет вложенных кавычек, интерпретатор показывает сроку в одиночных кавычках.

```
>>> "Язык "Питон"
'Язык "Питон'
```

б) Многострочные литералы.

Для удобства чтения текст может быть разбит на отдельные строки. Если его нужно представить литералом, то применяется другой синтаксис: многострочный литерал ограничивается с двух сторон *тремя* одиночными или тремя двойными кавычками (*triple-quoted strings*).

Внутри текста многострочного литерала допускаются как одинарные, так и двойные кавычки. Например,

Инструкция:	*Результат:*

`>>> print("""Язык "Питон"`
`версии`
`'3.5'`
`""")`
	`Язык "Питон"`
	`версии`
	`'3.5'`

Экранирование и escape-последовательности

Таблица ASCII содержит коды служебных символов, например, перевод строки или табуляция, которые используются в тексте наряду с буквами и другими символами.

На клавиатуре для служебных символов не предусмотрено специальных знаков. Поэтому, когда служебный символ нужно поместить в обычный текст, используются специальные комбинации символов, называемые *escape-последовательностями*.

Каждая escape-последовательность содержит определенную букву. Например, для символа "новая строка" (**n**ew line) это `n`, а для символа табуляции (**t**abulation) — `t` . Кроме того, чтобы показать, что буква используется в контексте служебного символа, перед ней нужно поставить защитный "экран" — символ обратного слеша (`\`). Поэтому для задания перевода строки в текст добавляется escape-последовательность `\n`, а для табуляции — `\t`.

Экранирование применяется также *для других целей*.

а) В escape-последовательностях символ "обратный слеш" играет служебную роль. Когда он встречается в тексте как обычный символ, его нужно экранировать — `\\`.

б) Экранирование вложенных кавычек позволяет использовать внутри текста строкового литерала тот же тип кавычек, что и ограничивающие его (одинарные с одинарными или двойные с двойными): `"Система \"Windows\" установлена"` .

в) Escape-последовательности позволяют задать в строке обычные печатаемые символы с помощью их числовых кодов. Коды могут быть заданы *тремя* 8-ричными цифрами, *двумя* 16-ричными или в двухбайтовом формате Unicode (два раза по две 16-ричные цифры).

Например, строку из латинских букв `'ABC'` можно представить так:

```
>>> '\101\102\103'          # 8-ричные коды из трех цифр
>>> '\x41\x42\x43'          # 16-ричные коды из двух цифр
>>> '\u0041\u0042\u0043'    # Unicode, четыре 16-ричные цифры
```

Часто используемые escape-последовательности приведены в справочной информации в конце текста.

Замечание

Некоторые escape-последовательности используются должным образом, только когда программа запускается как приложение из операционной системы. Например, звуковой сигнал системного динамика (beep, код `0x07`), запрограммированный с помощью escape-последовательности `print('\a')`, не звучит при выполнения кода в среде разработки.

В Питоне можно получить код любого символа в десятичной системе помощью функции **`ord`**`()`. Кроме того, есть функции **`oct`**`()` и **`hex`**`()`, которые позволяют переводить числа из десятичной системы в восьмеричную или шестнадцатеричную/ Их можно применять к десятичным кодам символов:

```
>>> hex(ord('&'))
'0x26'
```

ЗАДАНИЕ 1 (*Строковые литералы и экранирование*)

1. Поместить информацию о количестве дней в феврале в високосный и невисокосный годы в *одну* символьную строку. С помощью функции `print` информация выводится в две строки в виде:

```
високосный      29
невисокосный    28
```

Чтобы разделить на экране пустым пространством слова и числа, в строке между ними вставляется символ табуляции, а перед словом "`невисокосный`" — символ перевода строки.

2. В командной строке IDLE выполнить инструкцию

```
>>>"""Язык "Питон"
версии
'3.5' """
```

Объяснить, каким образом интерпретатор преобразовал строку?

Чем результат отличается, от результата приведенного ранее примера с выводом той же строки с помощью функции `print`?

3. В задании исследуются результаты вывода на экран литерала, содержащего путь к папке ОС Windows (т.е. с обратным слешем как разделителем элементов пути).

а) Выполнить инструкцию

```
>>> "c:\docs\new\text"
```

Как интерпретатор преобразовал строку? Почему экранирование выполнено только в одном случае?

б) Поместить ту же строку в функцию `print` и выполнить инструкцию:

```
>>>print("c:\ docs\new\text")
```

Как можно объяснить полученный результат? Что нужно сделать, чтобы результат стал ожидаемым?

4. Найти, какие шестнадцатеричные коды в таблице Unicode соответствует заглавной и строчной буквам `"A"` и `"a"` кириллицы. Сформировать из шестнадцатеричных кодов строку, которая будет выводиться функцией `print` как `'Веб'`.

Простейшие операции со строковыми литералами

В языке Питон используется универсальное понятие последовательности как коллекции объектов, упорядоченных по их позиции.

Строки — это один из типов *последовательностей*. Элементы строки проиндексированы (пронумерованы). Каждый символ строки индексируются в порядке слева направо. Так как индексы имеют смысл смещений от начала строки, то индексация начинается *с нуля*.

Есть вторая система нумерации, которая позволяет рассматривать строку *в обратном порядке*. Для этой нумерации использует отрицательные значения, причем индекс `-1` соответствует последнему символу строки, `-2` — предпоследнему и т.д.

прямая нумерация	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	П	**р**	**и**	**в**	**е**	**т**	**,**		**П**	**и**	**т**	**о**	**н**	**!**
обратная нумерация	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

а) *Доступ к отдельным символам*.

К каждому символу строки можно получить доступ с помощью *выражения индексирования*. Оно строится из литерала (или имени переменной) и целого значения индекса взятого в квадратные скобки: `строка[n]`, где `n` — позиция символа.

При *положительных* значениях индексов позиция отсчитывается слева направо *от нуля*. *Отрицательные* значения индекса применяются, чтобы просматривать строку справа налево (т.е. позиция отсчитывается от конца строки). Последний символ строки всегда получает индекс `-1`.

Значением выражения индексации является *строка*, состоящая из одного символа. Поэтому с формальной точки зрения символьные строки — это не последовательности отдельных символов, а последовательности простых строк, состоящих из одного символа.

Например,

```
>>>"Привет, Питон!"[0]
   'П'
```

```
>>>"Привет, Питон!"[-1]
    '!'
```

```
>>>'Привет, Питон!'[6]
    ','
```

Если значение индекса задано *нецелым* числом, то возникнет ошибка:

TypeError: string indices must be **integers**

Если значение индекса выходит за пределы строки, то ошибка другая:

IndexError: string index **out of range**

б) *Вычисление длины строки*.

Длина строки — это количество символов, из которых состоит строка (без учета внешних кавычек). Длину строки вычисляет функция `len` (от англ. `length`):

```
>>> len("abc")
3
```

в) *Конкатенация*.

Конкатенация (*лат*. сцепление) — это операция "склеивания" строк в одну общую. В Питоне есть несколько приемов для объединения строк.

- *Использование оператора конкатенации*.

Для объединения строк можно использовать оператор конкатенации, который задается символом +. Его операндами могут быть как литерал, так и переменных, хранящие строки.

При конкатенации никаких *символов-разделителей* между строками не добавляется. Когда программисту нужно между склеиваемыми строками вставить какой-то разделитель, например, пробел, то разделитель приходится встраивать в выражение конкатенации как дополнительную строку.

- *Неявная конкатенация*.

Если в строке инструкции соседствуют строковые *литералы*, между которыми нет никаких символов, *кроме пробелов*, то эти литералы будут автоматически объединены в один.

```
>>>"И.И."    "Иванов"
'И.И.Иванов'
```

Этот прием называется неявной конкатенацией. Так как строки "склеиваются" без явного использования оператора +, конкатенация называется *неявной*.

г) *Многократное повторение фрагмента строки (умножение)*.

Если в строке нужно повторить одну и ту же подстроку, то можно воспользоваться операцией конкатенации (т.е. "сложением").

Например, выражение `'Ура!'+'Ура!'+'Ура!'` даст строку `'Ура!Ура!Ура!'`.

Но, как и в случае числами, многократное сложение с одним и тем же слагаемым можно заменить умножением. Т.е использовать выражение `строка*n`, где `n` — число, задающее количество повторений.

Например, `'Ура!'*3`, повторит результат `'Ура!Ура!Ура!'`, полученный ранее конкатенацией.

ЗАДАНИЕ 2 (*строковые литералы*)

1. Какое значение вернет вызов функции `len("a\nb\tc")` и почему?

2. Что вернет выражение `"a\nb\tc"[-2]` и почему?

3. В выражении выполняется конкатенация строк `"Питон"` и `"язык программирования"`. Как нужно сформировать выражение (не трогая исходные строки), чтобы в результате была сформирована строка `"Питон - язык программирования"`.

4. Происходит ли неявная конкатенация литералов, если они имеют разные кавычки-ограничители?

Логический тип данных

Логический, или булев (`bool`) тип, имеется практически в каждом языке программирования. Это тип данных принимающих два возможных значения, которые в Питоне рассматриваются как константные объекты, обозначаемые литералами `True` (истина) и `False` (ложь).

Булев тип был добавлен в Питон с целью сделать код яснее. Как и во многих других языках, реально логический тип данных реализуется через целочисленный. Значение *истина* полагается равным `1`, за значение *ложь* — равным `0`.

Для хранения данных булева тип достаточно использовать один бит. Но обычно для них используется целый байт, так как в этом случае обеспечивается более быстрое обращение к памяти. Если данные имеют истиностную природу, то литералы `True` и `False` отражают её более явно, чем числовые значения `1` и `0`. Поэтому Питоном при выводе логических данных используются литералы.

```
>>> True
True
```

Вне логического контекста значения `True` и `False` интерпретируются, как имена целочисленных констант `1` и `0`. Например, это происходит в арифметических выражениях:

```
>>>True+2
3
```

Операторы сравнения

Операторы сравнения используются для проверки условий. В большинстве языков программирования, включая Питон, для них применяются следующие обозначения:

= =	—	равно,
!=	—	не равно,
>	—	больше,
<	—	меньше,
>=	—	больше или равно,
<=	—	меньше или равно.

Замечание.

Распространенной ошибкой в записи операций сравнения является использование вместо == (сравнение) знака = (присваивание). Другой тип ошибки — нестрогие неравенства *не могут* быть записаны с помощью комбинаций =< или =>. Также в версии 3 *не поддерживается* оператор "не равно" в виде < > .

Результатами вычисления выражений с операторами сравнения являются истиностные значения. Например, при проверке значений на совпадение:

```
>>> 5 == 5 # операнды с равными значениями
    True
>>> 5 == 6 # операнды с неравными значениями
    False
```

В первом выражении два операнда имеют одинаковые значения и поэтому результатом выражения будет `True`, а во втором — разные, поэтому получено `False`.

Сравнения могут проводиться не только для чисел, но практически для любых типов данных. Например,

```
>>> True == False
    False
>>> None==None  # None — фиктивное значение, "нет значения"
    True
```

Сравнение строк

При сравнении строк используются коды символов.

Если строки проверяются на совпадение, то, как только в одной из сравниваемых пар символы не совпадут, результат будет `False`.

```
>>> "bcd" == "bca"
False
```

При проверке на неравенство, итоговый результат определяется отношением между первой парой *несовпадающих* символов:

```
>>> "bcae" > "bcda"  # значение кода 'a' меньше, чем 'd'
False
```

Цепочки сравнений

Иногда проверяемое условие основывается на сочетании нескольких проверок. Например: "`количество повторений больше трех, но не больше 7`".

В традиционных языках программирования сложные условия формируются из простых проверок, объединяемых логическими связками. Например, "`больше 3`" **и** "`не больше 7`".

В Питоне можно создавать *цепочки сравнений*, когда в одном выражении участвуют несколько операторов сравнения:

а)	б)
`>>> 3 < 5 <=7` `True`	`>>> 3 < 5 > 4` `True`

Хотя пример б) будет успешно выполнен, пользоваться в одном выражении разнонаправленными проверками не рекомендуется.

ЗАДАНИЕ 3 (*Операторы сравнения*)

1. Выполнить сравнение символов 'a' из латинского и русского алфавитов, которые внешне выглядят одинаково, но имеют разные коды:

```
>>> 'a'<'a'
```

С помощью функции `ord` получить значения соответствующих им кодов и объяснить результат.

2. Выяснить, какая из двух строк будет считаться "большей", если вторая строка повторяет первую, но в конце имеет еще дополнительные символы.

3. Проверить, можно ли сравнивать логические константы с числами?

Логические выражения

Для формирование логических выражений в Питоне предлагается три базовых булевых оператора: `and` ("и"), `or` ("или") и `not` ("не").

а) *Оператор* `not`.

Унарный оператор `not` имеет самый высокий приоритет. Он применяется к одному операнду и меняет истиностное значение на противоположное (этим похож на арифметический оператор смены знака '–') :

а)	б)	в)
`>>> not True` `False`	`>>> not False` `True`	`>>> not not True` `True`

Результат применения оператора `not` всегда показывается Питоном с помощью логического литерала.

б) *Оператор* `and`.

Это бинарный оператор, который определяет логическую операцию "`и`", дающую значение `True` только в том случае, если оба операнда имеют значение "`истина`".

Оператор имеет приоритет, более низкий, чем `not`, но более высокий, чем `or`. Поэтому в следующем примере скобки необязательны:

а)	б)
`>>> (not False) and True` `True`	`>>> not False and True` `True`

в) *Оператор* `or`.

Это бинарный оператор, который определяет логическую операцию "`или`", дающую значение `False` только в том случае, когда оба операнда имеют значение "`ложь`".

Из трех логических операторов `or` имеет самый низкий приоритет. Поэтому при его использовании часто для изменения порядка вычисления выражения приходится использовать круглые скобки:

а)	б)
`>>> not False or True` `True`	`>>> not (False or True)` `False`

Числа и строки как операнды логических выражений

В выражениях с логическими операторами *истиностные значения* могут приписываться интерпретатором числам, строкам и более сложным объектам данных. Это полезно, потому что во многих случаях позволяет сделать код проверки условий более компактным.

Например, инструкция программы звучит как "`если строка не пустая, то ...`". Для буквальной проверки такого условия нужно сначала вычислить длину строки, а затем сравнить полученное значение с нулем: `len("123")!= 0`.

Однако в этом нет необходимости, так как непустая строка в контексте логического выражения интерпретируется как `True`. Например,

```
>>> not "asbcd"
False
```

Неформально общее правило такой интерпретации можно сформулировать следующим образом: "`пустые`" значения данных, интерпретируются как `False`, а "`непустые`" — как `True`.

Под "`пустыми`" нужно понимать нулевые числовые значения, пустые строки, псевдозначение `None` (обозначает отсутствие какого-либо значения), и др.

Сокращенные вычисления логических выражений

Когда Питон вычисляет выражения с операторами `and` или `or`, операции выполняются в порядке *слева направо*.

Если в выражении с оператором **`or`** *логической интерпретацией значения* первого операнда является `True`, то этого достаточно, чтобы сделать вывод об истинности всего выражения. Второй операнд вычислять не нужно.

Аналогично, когда при вычислении выражения с оператором **`and`** значение первого операнда *интерпретируется* как `False`, ложным будет *логический* результат всего выражения (без вычисления второго операнда).

Это дает возможность, применяя *сокращенные вычисления логических выражений*, ускорить их обработку.

Важно: при вычислении выражений с операторами `or` или `and` результатом всегда будет значение *последнего вычисленного операнда*.

Из-за сокращенных вычислений этот операнд *необязательно* является последним в выражении. Более того, хотя выражение логическое, оно *необязательно* имеет логический тип. Это демонстрируют следующие примеры.

```
>>> len("abc") and len("abcde")>=5
True                    # это результат вычисления len("abcde")>=5
>>> len("abcde")>=5 and len("abc")
3                       # это результат вычисления len("abc")
>>> len("abc") and "abcde"
'abcde'                 # это результат вычисления "abcde"
>>> (12-4*3) and 7>4
0                       # это результат вычисления 12-4*3
```

Логические операторы и условные выражения

Выражение, которое в зависимости от выполнения или невыполнения некоторого условия имеет разные значения, называется условным.

В Питоне есть специальная конструкция, позволяющая записывать *условные выражения* (не путать с условным оператором с теми же служебными словами):

`знач_true` **`if`** `условие` **`else`** `знач_false`

При успешном выполнении условия всё выражение получит значение `знач_true`, а при невыполнении — `знач_false`.

Например, в следующем примере условие не выполнено:

```
>>> 10 if len("abc")>5 else 20
20
```

Так как при вычислении выражений с `or` и `and` результатом является значение одного из *операндов*, то с их помощью можно смоделировать условные выражения:

`операнд-условие` **`and`** `результат`$_1$ **`or`** `результат`$_2$

В зависимости от истинности операнда-условия результатом вычисления выражения будет либо значение `результат`$_1$, либо `результат`$_2$.

Единственное ограничение — чтобы выражение работало, как условное, `результат`$_1$ и `результат`$_2$ должны интерпретироваться как истинные.

ЗАДАНИЕ 4 (*Логические выражения*)

1. Выполнить сравнение символов 'a' из латинского и русского алфавитов (внешне выглядят одинаково):

```
>>> 'a'<'a'
```

С помощью функции `ord` получить значения соответствующих им кодов и объяснить результат.

2. Выполнить инструкции `1.0` **`or`** `1/0` и `1/0` **`or`** `1.0`.

Учитывая, что `1.0` трактуется как `True`, с математической точки зрения в обоих случаях результатом должна быть "истина".

Объяснить, почему различаются результаты выполнения инструкций?

3. Вычислить выражение `"" and 1 or "123"` и объяснить результат.

4. Вычислить выражение `"102"[1] and "102"[-1]` и объяснить результат.

5. Когда будет истинно значение выражения `n%2==0 and n%3==0` ?

Вопросы для самоконтроля

1. Из каких элементов строятся тексты программ? Что понимается под литералом?
2. Какие типы числовых литералов рассматриваются в Питоне?
3. Какие типы строковых литералов имеются в Питоне? В чем различия в их записи? Что называется пустой строкой?
4. Как получить код символа в десятичной системе и преобразовать его в восьмеричную и шестнадцатеричную системы счисления?
5. Приведите примеры escape-последовательностей. Для каких целей применяется экранирование символов?
6. Какие есть варианты для размещения кавычек внутри текста литералов?
7. В чем особенность символьных строк как последовательностей?
8. Как получить доступ к отдельному символу строки?
9. Как вычисляется длина строки?
10. Что понимается под конкатенацией строк? Какой оператор используется для выполнения конкатенации? Когда выполняется неявная конкатенация?
11. Как значения булева типа связаны с числовыми?
12. Какие операторы используются для сравнения значений и какие результаты получаются при их выполнении?
13. Что понимается под цепочкой сравнений?
14. Как числа и строки интерпретируются в контексте логических выражений?
15. В чем идея сокращенного вычисления логических выражений? Что можно сказать о результатах таких вычислений?
16. Что понимается под условным выражением? Как записываются условные выражения (с помощью служебных слов `if-else`)? как условные выражения моделируются с помощью операторов `and-or`?
17. С помощью какой функции можно получить код символа в десятичной системе?

Справочная информация

Escape-последовательности

Символы	Назначение
`\\`	Обратный слеш \
`\'`	Одиночная кавычка
`\"`	Двойная кавычка
`\a`	Звук системного динамика Bell (BEL)
`\n`	Новая строка. Перемещает курсор в начало следующей строки LF line feed, ASCII code 10 (0x0A).
`\t`	Символ табуляции, горизонтальный отступ. Перемещает курсор вправо на одну позицию табуляции
`\b`	Backspace, стереть последний символ
`\ooo`	Код символа в восьмеричной системе (три цифры)
`\xhh`	Код символа в шестнадцатеричной системе (две цифры hh)
`\uhhhh`	Код символа в двухбайтовой кодировке Unicode (четыре шестнадцатеричные цифры hhhh)

Старшинство операторов

В таблице операторы приведены в порядке возрастания их приоритетов. В одной строке операторы с равным приоритетами.

Операторы	Смысл операции
`==, !=`	Операторы равенства
`<=, <, >, >=`	Операторы сравнения
`+, -`	Сложение и вычитание
`*, /, //, %`	Умножение, деление, остаток
`+x, -x`	Определение и смена знака (унарные)
`**`	Возведение в степень
`x.свойство`	Обращение к свойству