ЛАБОРАТОРНАЯ РАБОТА

Тема: ***Явные преобразования типов. Присваивание и хранение данных***

Цель работы: Познакомиться с явными преобразованиями типов данных, приемами организации простейшего диалога, оператором присваивания и особенностями применения переменных в Питоне.

Отчет: Отчет представляет собой документ Word, в который последовательно из окна интерпретатора копируются команды и результаты выполнения для каждого задания.

Содержание

Явные преобразования типа

Так как Питон язык с сильной типизацией, в бинарных операциях операнды должны иметь один и тот же тип. Если типы разные, то значение одного из операндов должно быть преобразовано к типу другого операнда.

Поэтому при обработке выражений *неявные преобразования* (автоматические, без ведома программистов) выполняются только в отдельных случаях. Так, в выражении 1+2.0 первый операнд неявно будет преобразован к вещественному типу (1**.0**) и затем выполнено сложение для вещественных чисел (результат — 3**.0**).

Когда у операндов различие типов более серьезное, то у Питона возникают подозрения, что это результат ошибки программиста, поэтому неявные преобразования не выполняются.

Например, при сложении числа и строки, содержащей запись числа, будет сгенерировано сообщение о невозможности выполнить операцию:

```
>>> 1+'23'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Чтобы вычислить выражение, необходимое преобразование типа операнда должно быть предусмотрено в коде. Операции преобразования типа, включенные в программу, называются *явными*.

Питон имеет набор встроенных функций для выполнения явных преобразований типа.

а) *Преобразование к целому типу* (функция ***`int`***`()`).

Функция `int` преобразует свой аргумент в целое числовое значение, если ей передаются либо вещественное число, либо строка с текстовым представлением числа в десятичной системе:

```
>>> 1+int('23')
24
>>> int("0x1A")  #пытаемся преобразовать представление 16-ричного числа
ValueError: invalid literal for int() with base 10: '0x1A'
```

Ошибка всегда возникает, если строка имеет нечисловое содержание:

```
>>> int("2a")
ValueError: invalid literal for int() with base 10: '2a'
```

Когда `int` преобразует числа с точкой в целые, но дробная часть *отбрасывается* (округления нет):

б) Преобразование *к вещественному типу* (функция ***float***()).

Когда это возможно, функция `float` преобразует свой аргумент в число с точкой:

```
>>> float(32)
32.0
```

```
>>> float("3.14159")
3.14159
```

```
>>> float(True)
1.0
```

в) Преобразование *в строковое представление* (функция ***str***()).

Функция `str()` выполняет преобразование значений различных типов (числовых, булевых и других более сложных) к строковому виду

```
>>> str(32)
'32'
```

```
>>> str(3.14149)
'3.14149'
```

```
>>> str(False)
'False'
```

г) *Преобразование к булеву типу* (функция ***bool***()).

Функция `bool()` выполняет приведение данных к булеву типу, если это возможно. Она преобразует "пустые" объекты (целые и вещественные нули, пустые строки и др.) в `False`, а остальные — в `True`.

```
>>> bool('')
False
```

```
>>> bool(0.0)
False
```

```
>>> bool("строка")
True
```

Создание интерактивных сценариев (функция `input`)

Часто работу программы удобно реализовать как диалог. В нём пользователь может вводить данные, а программа обрабатывает их и отвечает выводом результатов.

Для чтения вводимой пользователем информации в Питоне имеется функция `input()`. При её выполнении программа переходит в режим ожидания и находится в нем пока пользователь не подтвердит завершение ввода нажатием клавиши `Enter`.

Функция возвращает введенные пользователем данные в виде *строки*. В этом просто убедиться, выполнив следующие две инструкции:

```
>>> s = input()
123  Enter
>>> type(s)
<class 'str'>
```

Если программе требуется не строка, а данные другого типа, то потребуется явно преобразовать возвращаемое `input()` значение к нужному типу.

Функция `input()` всегда возвращает строку. Если пользователь не будет ничего вводить, а просто нажмет клавишу `Enter`, то функция вернет *пустую строку*. Если привести возвращаемое функцией `input()` значение к логическому типу, только в одном единственном случае будет `False`. В каком?

Чтобы пользователь понял, что программа ждет от него данных, и не путался, какие данные нужно ввести, нужна подсказка. Текст подсказки можно задать в (необязательном) аргументе функции `input()` и он будет выведен на консоль:

```
>>> input('Введите год: ')
Введите год: 2016
'2016'
```

ЗАДАНИЕ 1 (*Явные преобразования типа*)

1. Преобразовать к целочисленному типу значения `3.9999999`, `2.5` и `'-2.3'`, `True`. Объяснить результаты.

2. Преобразовать к вещественному типу значения `999`, `'-1.25e-3'`, `True`, `'False'`. Какие выводы можно сделать из этих примеров в каждом случае?

3. Убедиться, что комплексное число *нельзя* явно преобразовать к целому или вещественному типу.

4. Описать одной инструкцией следующие действия: в диалоге с помощью функции input пользователь вводит два целых числа и в этом же выражении подсчитывается их сумма. Результат выполнения инструкции должен выглядеть так:

```
Первое слагаемое: 3
Второе слагаемое: 5
8
```

5. Описать одной инструкцией условное выражение, в котором в роли операнда-условия используется функция `input()`, а результаты — строки `"Да"` (для `True`) и `"Нет"` (для `False`). Проверить выполнение инструкции. Что нужно ввести пользователю, чтобы был получен ответ `"Нет"`?

Оператор присваивания

Каждое значение (число, строка или данные любого другого типа) хранится Питоном в памяти как объект. Значения получаются как результат вычислений выражений. Чтобы созданный выражением объект данных далее можно было использовать в программе, применяются переменные.

Для создания переменных никакие специальные описания в программах не требуются.

а) *Простое присваивание.*

Новые переменные возникают в момент присваивания (*англ.* `assignment`) им значений:

```
var = expr
```

Оператор присваивания `'='` *связывает* (*англ.* `bind`) имя переменной `var` со значением выражения `expr`. Под *связыванием* понимается сохранение в переменной ссылки на значение (т.е. адреса объекта данных), а не самого значения.

Порядок выполнения инструкции присваивания:

- *шаг 1*: сначала вычисляется выражение `expr`,
- *шаг 2*: значение `expr` сохраняется в памяти как новый объект данных,
- *шаг 3*: создается переменная `var`,
- *шаг 4*: переменная *связывается* со значением (объектом данных) — в ней запоминается адрес объекта данных.

Замечания.

1. Если значение, полученное при вычислении `expr`, уже хранится в памяти, то новый объект на шаге 2 *может* не создаваться, а для связывания использоваться существующее значение.

2. Если переменная `var` уже была создана ранее, то при повторном присваивании её создавать заново не нужно — шаг 3 пропускается. Переменная просто будет связана с другим объектом данных (шаг 4).

Замечание.

Хотя реально переменные хранят ссылки на значения, это особенность *внутренней* организации памяти, которая в Питоне, как языке высокого уровня, должна быть скрыта от программиста.

При вычислении выражений идентификаторы переменных автоматически заменяются значениями, с которыми связаны эти переменные. В результате всё выглядит так, как если бы переменная действительно хранила само значение. Поэтому, когда используется термин

"значение переменной", то всегда подразумевается связанное с переменной значение (объект данных), а не ссылка.

Если переменной значение еще не было присвоено, то она просто не существует. Это значит, что до выполнения присваивания любая попытка использовать значение переменной обернется ошибкой:

```
>>> print(z)
NameError: name 'z' is not defined
```

б) ***Групповое присваивание.***

Выражение `expr` может иметь любую сложность. В простейшем случае это константа или переменная. Если переменной `var` присваивается другая переменная, то после выполнения инструкции *обе переменные* будут ссылаться *на одно значение*.

Одной инструкцией *одно и тоже значение* может быть присвоено сразу нескольким переменным (групповое присваивание):

```
var3 = var2 = var1 = expr;
```

Порядок действий следующий:

- сначала переменной `var1` присваивается значение выражения `expr`,
- затем выполняется присваивание `var2 = var1`,
- и т.д.

```
>>> a = 10
>>> c = b = a
>>> print(c)
10
```

Комбинированные операции. Неизменяемость значений

Есть много случаев, когда в программах значение переменной "накапливается" постепенно.

Например, если переменная `counter` будет использоваться как счетчик некоторых событий, то при создании она должна получить значение `0`, а затем при каждом появлении события к счетчику (*к себе*) добавляется единица: **`count = count + 1`**.

Похожим способом можно вычислять факториал натурального числа. Сначала переменная `fact` получает значение `1`, а затем новые значения она последовательно получает путем умножения *себя* на очередное натуральное число:

```
fact = 1
fact = fact *2
fact = fact *3
...
```

В обоих случаях у инструкций общая особенность — новое значение переменной вычисляется через её предыдущее значение.

Чтобы сделать запись таких инструкций более лаконичной, в выражениях можно применять операторы *комбинированного присваивания* (из двух символов, с любым бинарным оператором):

`+=` `-=` `*=` `/=` `//=` `%=` `**=` и др.

Например, `count` **`+=`** `1` или `fact` **`*=`** `3`.

Для комбинированных операций используется обычный порядок вычислений, определенный для присваивания. Поэтому, если справа стоит выражение, то вычисления начнутся с него.

Операцию `+=` и `*=` можно применять не только к переменным с числовыми, но и со строковыми значениями.

Для строк операция `+=` означает конкатенацию переменной со строкой, находящейся в правой части.

Операция `*=` подразумевает многократную конкатенацию строки с самой собой. Она имеет смысл, если справа стоит целое положительное число (при отрицательном в в результате получится пустая строка).

В Питоне числа и строки относятся к *неизменяемым объектам* (*англ.* `immutable`) данных. Т.е. после создания эти объекты изменить нельзя. Например, не получится в строке заменить один символ на другой.

Поэтому каждый раз при выполнении операции комбинированного присваивания

- на основании текущего значения переменной вычисляется и создается объект нового значения и
- в переменной запоминается ссылка на это новое значение.

При этом объект со значением, на который раньше ссылалась переменная, сохраняется.

Идентификатор объекта данных

Когда создается новый объект данных, для него формируется *уникальный числовой* идентификатор, который остается неизменным всё время существования объекта.

В реализациях Питона в роли идентификатора обычно используется адрес области памяти, где хранятся данные.

Получить идентификатор объекта данных можно с помощью функции

`id(`*`object`*`)`.

Если аргументом функции является выражение, то сначала выполняется вычисление выражения, а затем функция применяется к результату вычислений.

Если аргумент функции `id` — переменная, то функция возвращает внутренний идентификатор *значения*, с которым связана переменная.

```
>>> id(1)
1657512384
```

```
>>> id("строка")
47337624
```

```
>>> x='abc'
>>> id(x)
35385784
>>> id('abc')
35385784
```

Присваивание, т.е. связывание переменной с объектом, можно представить как запоминание в переменной внутреннего идентификатора объекта данных.

Очевидно, что с одним объектом данных (значением) могут быть связаны несколько переменных.

Анализ значений идентификатора данных удобен при отладке программ и позволяет понять принципы хранения данных в Питоне.

ЗАДАНИЕ 2 (*Внутреннее представление данных и переменных*)

Чтобы повысить быстродействие программ, при загрузке Питона автоматически создаются объекты данных для небольших чисел (зависит от системы, например, от -5 до 256) и коротких строк (пустая строка и, возможно, другие).

1. Выполнить следующие действия:

   а) присвоить переменной `x` значение `1000`;

   б) присвоить переменной `y` переменную `x`;

   в) получить `id` для обеих переменных;

   г) присвоить `x` новое значение `1001`;

   Для переменных `x` и `y` сформировать и вывести две текстовые строки вида:

   `x =` `значение_x` **`id =`** `значение_id_x`
   `y =` `значение_y` **`id =`** `значение_id_y`

Для построения строк применить операции конкатенации и преобразования типа.

Можно строить строки и вызывать `print` для вывода информации о каждой переменной отдельно.

Можно подготовить одну общую строку с информацией об обеих переменных, включающую символ перевода строки, и вызвать `print` один раз.

Объяснить результаты.

2. Выполнить инструкции

```
>>> x=2
>>> x*=2+3
```

Выяснить итоговое значение переменной `x` и объяснить, как был получен результат.

3. Переменная `x` получает значение `81`, затем с помощью комбинированного оператора вычисляется её новое значение —утроенное начальное.

Получить и сравнить `id`:

а) переменной `x` после первого присваивания,

б) переменной `x` после изменения значения,

в) значения `81`,

г) значения `243`.

Объяснить результаты.

Повторить все действия для случая, когда `x` получает значение `512` (соответственно, утроенное — `1536`).

Объяснить, чем отличаются результаты второго эксперимента от первого.

4. Переменной `next_day` (т.е. следующий день) присваивается строка

`"После 28 февраля будет ",.`

Переменной `year` (т.е. год) присваивается значение `2016`.

Строка `next_day` должна быть дополнена текстом либо `"1 марта"`, если год невисокосный, либо `"29 февраля"`, если год високосный. Для этого использовать оператор комбинированного присваивания `+=` (как определить, что год високосный или нет?).

Выбор дополняющей подстроки реализуется с помощью условного выражения, которое помещается в правую часть оператора присваивания.

Убедиться, что для високосного 2016 года получен правильный результат. Повторить все действия, задав невисокосный 2017 год.

Вопросы для самоконтроля

1. В чем различие явных и неявных преобразований типа?

2. Как показать, что Питон относится к языкам со строгой типизацией?

3. Какие функции используются для явного преобразования к целочисленному, вещественному, строковому и булеву типам?

4. К каким данным применима операция приведения к целочисленному значению? Что происходит при преобразовании вещественного числа к целому типу?

5. Для чего используется функция `input`? Какие есть варианты её синтаксиса? Что возвращает функция `input`?

6. Как получаются новые объекты данных (значения)? Когда создаются новые переменные?

7. В чем суть выполнения присваивания? Что понимается под связыванием?

8. Каков порядок выполнения операции присваивания?

9. Что происходит, если в программе делается попытка использовать значение еще не созданной переменной?

10. Как выполняется групповое присваивание?

11. Какой смысл имеют для строк операции `+=` и `*=`?

12. Что понимается под неизменяемыми объектами?

13. Каков порядок выполнения операций комбинированного присваивания?

14. Какие комбинированные операции присваивания применимы к строкам?

14. Что обычно используется в роли внутреннего идентификатора объекта данных?

15. Что возвращает функция `id()`, когда она применяется к данным и к переменным?