ЛАБОРАТОРНАЯ РАБОТА

Тема: Применение списковых выражений

Цель работы: Освоить технику генерации и фильтрации списков с помощью списковых выражений

Содержание

Списковое выражение

На практике часто возникает необходимость выполнить однотипную обработку каждого из членов некоторой последовательности. Например, необходимо вычислить значения некоторой функции для заданного набора значений аргумента (построить таблицу значений, задача табуляции функции).

Это типичная ситуации, в которой удобно использовать оператор цикла. Если требуется получить список значений синуса для углов π/6, π/4 и π/3 можно использовать следующий код:

```
import math                 # импортировать математический модуль
result=[]                   # сначала результирующий список пуст
for x in [math.pi/6, math.pi/4, math.pi/3]:
    result.append(math.sin(x))     # добавить очередное значение
print(result)
```

Результатом выполнения программы будет список

```
[0.49999999999999994, 0.7071067811865475, 0.8660254037844386].
```

В Питоне есть специальная высокоуровневая конструкция, позволяющая решение таких задач записывать более компактно. Конструкция называется *списковым выражением* (генератором списков или *списковым включением*, от *англ*. list comprehension — включение в списки).

Замечание

Эта конструкция является одним из вариантов реализации *отображения* (map) одной последовательности на другую.

Более точное описание спискового выражения.

- Имеется некоторая *исходная последовательность*, из которой поочередно выбираются значения для обработки.
- Каждое выбранное значение запоминается во вспомогательной *переменной*.
- Вспомогательная переменная (т.е. значение из исходной последовательности) участвует в *выражении*, описывающем способ вычисления элементов новой последовательности.
- Вся конструкция спискового выражения заключается в *квадратные скобки*:

```
[ выражение    for переменная in последовательность ]
```

Исходная *последовательность* может быть списком, кортежем или строкой символов. Результатом вычисления будет новый список, а обрабатываемая последовательность остается неизменной.

Операции, выполняемые при вычислении списковых выражений, нетрудно реализовать с помощью обычного цикла `for-in`. Поэтому конструкция списковых выражений кажется избыточной. Однако наглядна, так как ясно отражает смысл выполняемых действий и удобна, так как позволяет более компактно записать программный код. Поэтому конструкцию списковых выражений можно отнести к категории "синтаксический сахар".

Замечание.

Под «синтаксическим сахаром» понимается любой имеющийся в языке программирования синтаксический элемент или способ описания, который хотя и дублирует другой, имеющийся в языке, но является более удобным в использовании и/или более кратким.

ЗАДАНИЕ 1 (*Использование списковых выражений*)

1. Решить с помощью спискового выражения задачу вычисления значений синуса для трех углов $\pi/6$, $\pi/4$ и $\pi/3$ двумя способами: задав углы списком (квадратные скобки) или кортежем (круглые).

2. Решить с помощью спискового выражения задачу вычисления списка, состоящего из значений синуса и косинуса (т.е. вычисляемое выражение является списком!), найденных для каждого из трех углов $\pi/6$, $\pi/4$ и $\pi/3$.

Фильтрация элементов в списковых выражениях

Задача обработки исходной последовательности усложняется, когда её элементы должны обрабатываться выборочно — в формировании результирующего списка участвуют только те элементы, которые отвечают некоторому условию. Это условие обеспечивает "процеживание" элементов последовательности.

В результате некоторые элементы исходной последовательности, не прошедшие через "сито" проверки, будут отброшены. Поэтому в таких случаях говорят, что выполняется пошаговая *фильтрация* последовательности.

Конструкция списковых выражений в этом случае расширяется за счет добавления кода с проверкой условия:

```
[ выражение  for переменная in последовательность if лог.выражение ]
```

Если для очередного элемента условие выполнено, то вычисляется новое значение выражения, которое добавляется в результирующий список. Если условие не выполнено, то обработка элемента пропускается. Поэтому в итоговом списке может оказаться меньше элементов, чем в было исходном.

Например, необходимо оставить в числовой последовательности только положительные числа

```
>>> result=[ x   for x in [-1,2,-3,4,-5]   if x>0 ]
>>> result
[2, 4]
```

ЗАДАНИЕ 2 (*Применение спискового выражения для фильтрации*)

Задан список из нескольких элементов строк:

```
["первый", "2-ой", "третий", 4, [5], "последний"].
```

С помощью фильтрующего списочного выражения необходимо выбрать из этого списка только *слова* (строки из букв) и сформировать из них новый список.

При решении задачи нужно учесть, что элемент исходного списка попадает в результирующий, при *одновременном* выполнении двух условий:

- элемент является строкой,
- эта строка содержит слово (состоит только из букв).

Чтобы выполнить проверку нужно сформировать логическое выражение с оператором **and**, в котором

- с помощью функции-предиката `isinstance(elem, str)` проверяется, является ли элемент списка строкой, **и**,

- когда элемент является строкой, с помощью метода-предиката `elem.isalpha()` проверяется, представляет ли строка слово.

Так как в списке могут оказаться данные, не являющиеся строками, к которым нельзя применить метод `.isalpha()`, то порядок проверок в логическом выражении важен.

ЗАДАНИЕ 3 (*Проверка типа введенных данных — функция check_input*)

В заданиях 2 и 3 лабораторной работы 10 рассмотрены вспомогательные для программы "Палиндром" функции, обеспечивающие ввод и проверку допустимости введенной информации. Одна обеспечивала ввод одного слова, а вторая — списка слов.

Два варианта можно объединить в функции, способной проанализировать, что ввел пользователь — слово или список слов, и далее действовать в соответствии с этим.

Если вводится слово, то его можно поместить в список из одного элемента. Тогда дальнейшая обработка информации программой "Палиндром" всегда будет исходить из того, что был введен список.

Для первого этапа обработки информации необходимо создать функцию `check_input(data)`, которой через аргумент `data` передается строка с введенными пользователем данными.

Замечание

Применение функции `check_input()` в программе "Палиндром" будет рассмотрено в следующей лабораторной работе.

1. *Возвращаемое значение.*

Если данные допустимы, то функция возвращает их в виде списка.

Если данные недопустимы, то функция возвращает *пустой список*.

Для хранения возвращаемого значения в функции используется вспомогательная переменная `words`.

2. *Тело функции.*

Функция `eval`, которая будет применена для преобразований, не может обрабатывать пустые строки и пустые списки (происходит *ошибка*). Чтобы избежать этого сначала выполняется проверка, не являются ли введенные пользователем данные "пустыми".

Если (т.е. `if`) строка `data` — пустая (`''`) **или** (`or`) состоит из пустого списка (`'[]'`), то в теле условного оператора выполняется две инструкции: на экран выводится сообщение `'Пустые данные!'` и переменной `words` присваивается пустой список.

Иначе (т.е. `else`) нужно проверить, что ввел пользователь, список или строку.

Для этого используется вложенный (в тело `else`) еще один условный оператор.

Строка `data` будет содержать список, если первым символом строки является `"["` **и** одновременно (`and`) последним символом является `"]"`. Если соответствующее логическое выражение имеет значение `True`, то к `data` применяется функция `eval()` и результат сохраняется в переменной `words`.

Иначе `data` содержит обычную строку, из которой создаем список из одного элемента и запоминает его в переменной `words`.

Функция возвращает переменную `words`.

Алгоритм, реализуемый функцией, приведен в *Приложении 1*.

3. *Промежуточное тестирование* функции `check_input()`.

Тестирование программы — это процесс её испытания с целью выявить ситуации, когда поведение программы является неправильным. Тестирование может выполняться на разных этапах разработки. Это позволяет выявить возможные ошибки еще на ранних стадиях разработки. Если это не сделать, то при тестировании программного продукта в целом могут возникнуть сложности с локализацией ошибки (определением места ошибки).

При выполнении текущего задания реализована важная часть функции `check_input()`, которая будет использоваться как основа для добавления новой функциональности. Поэтому нужно с помощью промежуточного тестирования убедиться, что функция в нынешнем состоянии работает без ошибок.

Для этого необходимо проверить правильность выполнения функции `check_input()` при вводе:

- пустой строки (`Enter`),
- строки с пустым списком (`'[]'`),
- произвольной строки,
- произвольного списка.

4. Перед тем, как передавать список для проверки на принадлежность слов к палиндромам, нужно удалить из списка `words` все элементы, не являющиеся словами.

Для этого добавить перед оператором возврата инструкцию, в которой с помощью фильтрующего спискового выражения выполняется отбор из списка `words` только *слов*. Для этого нужно повторить инструкции, использованные при выполнении задания 2.

Итоговый список (результат вычисления спискового выражения) должен стать возвращаемым значением функции. Его можно запомнить в новой переменной. Но тогда придется менять ранее использованную инструкцию оператора возврата `return` **`words`**.

Поэтому сгенерированный список удобнее сохранить в той же самой переменной `words`, которая ссылалась на исходный список (объяснить, почему это возможно, в каком порядке выполняется операция присваивания?).

5. Проверить выполнение нового варианта функции `check_input()` при вводе:

- одного слова (строки, состоящей только из букв),
- строки, включающей буквы, цифры или другие символы,
- списка из слов,
- списка, состоящего из слов и строк с произвольными символами.

Вопросы для самоконтроля

1. Какую синтаксическую структуру имеет списочное выражение?
2. Какова роль отдельных элементов в синтаксической структуре списочного выражения?
3. Что в программировании понимается под "синтаксическим сахаром"? Почему списочное выражение можно охарактеризовать как "синтаксический сахар"?
4. Какие типы данных может представлять исходная последовательность, обрабатываемая списковым выражением?
5. Что понимается под фильтрацией элементов?
6. С помощью каких предикатов можно выполнить проверку, что элемент произвольного списка представляет слово? В каком порядке должны выполняться проверки и почему?
7. Какую синтаксическую структуру имеет списочное выражение, позволяющее обрабатывать элементы исходной последовательности выборочно?

8. Какие строки не могут быть обработаны функцией eval без ошибок?

9. Что понимается под тестированием программы?

10. Почему тестирование необходимо выполнять на разных этапах разработки, а не только в конце?

Приложение 1

Блок-схема алгоритма функции check_input()

Блок-схема *не включает* фильтрацию элементов списка с целью выделения из него слов (задание 3.4).