**ЛАБОРАТОРНАЯ РАБОТА**

***Тема***: ***Управление вводом информации***

***Цель работы***: Изучение методов ввода и валидации (проверки) данных.

***Содержание***



## Уточнение формальной постановки задачи

В лабораторной работе №7 была сформулирована следующая постановка задачи проверки строки на принадлежность к палиндромам.

*Приложение предназначено для анализа на принадлежность к палиндромам одного слова — символьной строки, состоящей только из букв. Буквы рассматриваются без учета их регистров.*

Ввод и проверку содержания введённой строки требованиям (только из букв). обеспечивала функция `get_word()`. Если введённая строка оказывалась пустой или содержащей посторонние символы, выводилось сообщение "`Ошибка ввода!!!`" и выполнение программы завершалось.

В лабораторной работе рассматриваются три варианта *функции для ввода* слов, которые, как предполагается, в дальнейшем должны анализироваться на принадлежность к палиндромам. Усовершенствованию кода для проведения собственно анализа слов будет посвящена следующая лабораторная работа.

**ЗАДАНИЕ 1** (*Дружелюбное поведение процедуры ввода слов*)

В случае неправильно введенной информации программа должна вести себя более дружелюбно по отношению к пользователю, не прекращать работу с сообщением об ошибке, а давать возможность повторить ввод.

Создается функция **`get_word_1`**`()`, отвечающая этому требованию. После успешного ввода слова функция возвращает это слово.

***1. Реализация повторных попыток.***

Если пользователь ввел недопустимую строку, то функция предлагает ему повторно вводить слово до тех пор, пока не будет получена строка, состоящая только из букв.

Для организации повторений используется цикл `while`. Чтобы управлять его работой, в функции `get_word_1()` определяется вспомогательная переменная `no_data` с начальным значением `True`, которое говорит о том, что требуемые данные еще не введены.

```
while no_data:
    инструкции для ввода и проверки слова
```

После того, как пользователь ввёл допустимое слово, переменная `no_data` меняет значение на `False` и циклические повторения завершатся.

В теле цикла выполняются следующие действия.

а) Пользователю предлагается подсказка "`Введите слово: `" и введенная им строка запоминается в переменной `word`.

б) В операторе `if-else` логическое выражение (условие) состоит из вызова для строки `word` метода-предиката `isalpha()`.

Если введена строка, состоящая только из букв (результатом применения предиката будет "истина"), то переменной `no_data` присваивается значение `False`.

Если введена недопустимая строка, то выводится сообщение "`Повторить ввод!`".

Функция должна возвратить введенное слово. При выходе из цикла оно будет находиться в переменной `word`.

***2. Тестирование функции.***

Проверяется правильность работы функции `get_word_1()`.

Для этого требуется подготовить вспомогательный код, где осуществляется вызов этой функция, а возвращаемое ею значение выводится на экран.

Проверить возвращаемый функцией результат при вводе пользователем:

– слова из букв,

– строки, включающей цифры и буквы, и

– пустой строки.

**ЗАДАНИЕ 2** (*Ограничение попыток ввода слова*)

Когда пользователь при повторных попытках постоянно вводит недопустимые строки, то в рассмотренном варианте функции `get_word_1()` цикл `while` фактически становится бесконечным. Это может происходить, например, когда пользователь не понимает, что от него требуется. Поэтому имеет смысл ограничить число попыток.

Разрабатывается новый вариант функции для ввода слов, в котором пользователю разрешено сделать не более фиксированного числа попыток.

Как только лимит исчерпан, он получает сообщение "`Превышен лимит попыток`!" и программа завершает работу.

***1. Управление количеством попыток.***

В функции **`get_word_2()`** определяется переменная `max_attempts` (т.е. максимальное число попыток), которой задается, например, значение `5`.

***2. Организация циклических повторений.***

Так как максимальное число возможных повторений задано, то можно воспользоваться циклом `for-in` с диапазоном, определяемым значением переменной `max_attempts` (*англ*. attempt — попытка).

```
for attempt in range(max_attempts):
    инструкции для ввода и проверки слова
```

В теле цикла выполняются следующие действия.

а) Пользователю предлагается подсказка "`Введите слово: `" и введенная им строка запоминается в переменной `word`.

б) В операторе `if-else` логическое выражение состоит из вызова для строки `word` метода-предиката `isalpha()`.

Если введенная строка состоит только из букв ("`истина`"), то с помощью оператора `break` выполняется выход из цикла.

Если строка недопустимая, то выводится сообщение "`Повторить ввод!`".

в) Когда пользователь не смог за разрешенное число попыток правильно ввести информацию, то необходимо вывести сообщение (вставить значение `max_attempts`)

"`Выполнено ... неудачных попыток. Выход из программы!`",

и затем завершить выполнение программы. Для этого, как и ранее, использовать вызов функции `sys.exit()` из импортируемого модуля `sys`.

Для вставки в строку значения `max_attempts` имеет смысл воспользоваться форматной строкой (%).

Действия, необходимые при исчерпании попыток, удобно поместить в блок `else` оператора `for-in` (см. справочную информацию).

***3. Тестирование функции.***

Проверяется правильность работы функции `get_word_2()`.

Для этого требуется подготовить вспомогательный код, где осуществляется вызов этой функции, а возвращаемое ею значение выводится на экран.

Проверить возвращаемый результат функции (задать `max_attempts=3`):

а) при вводе допустимого слова,

б) при последовательном вводе строки, включающей цифры, пустой строки и допустимой строки,

в) при превышающем лимит числа попыток вводе неправильных строк, например, пустых.

## Метапрограммирование. Функции `eval` и `exec`

*Метапрограммирование* — техника создания программ, которые в процессе своего выполнения тем или иным образом формируют фрагменты нового кода (которого не было в исходном тексте программы) и выполняют их.

В наиболее распространенной форме метапрограммирование реализуется, когда программа динамически формирует *строку*, содержащую код на языке программирования. Затем эта строка передается специальной функции, которая исполняет этот код.

В Питоне есть две функции, обеспечивающие возможность выполнения кода, заданного текстовой строкой.

Функция `eval` (от evaluate — вычислить) в качестве аргумента принимает строку, описывающую одно *выражение* Питона, вычисляет это выражение и *возвращает* полученное значение.

Функция `exec` (от execute — выполнить) принимает в качестве аргумента строку, описывающую *произвольный* программный код и выполняет его. Текст фрагмента программы может включать условные операторы, операторы цикла, определения функций и другие конструкции. Так как подавляющее большинство инструкций языка не имеет возвращаемого результата, логично, что `exec` не возвращает содержательный результат (возвращает фиктивное значение `None`).

*Пример.*

```
>>> new_code="s='Результат выполнения строки с кодом';print(s)"
>>> exec(new_code)
Результат выполнения строки с кодом
```

Простейшее применение функции `eval` — преобразование к естественной форме чисел, списков, кортежей и других данных, представленных в строковом виде. Она может использоваться как замена функций для преобразования строк в другие типы данных `int`, `float` и др. (но работает медленнее). Например,

```
>>> eval('1.5')
1.5
```

Предположим, что пользователь с помощью `input` ввел список чисел.

```
>>> word_list = input("Введите массив чисел: ")
Введите массив чисел: [1,2,3]
>>> word_list
'[1,2,3]'
```

Так как `input` всегда возвращает строки, то результатом ввода — не список, а строка, в которую помещен список. При вычислении содержания этой строки (т.е. списка) результатом будет сам список. Поэтому для преобразования пользовательского ввода в список достаточно применить к результату функцию `eval`:

```
>>> eval(word_list)
[1, 2, 3]
```

## Проверка принадлежности данных к заданному типу

Когда в программе с помощью `input()` вводятся данные, то они возвращаются в виде строки. Что находится в строке, можно определить с помощью методов-предикатов *строковых объектов*, таких как `.isalpha()` или `.isnumeric()`.

При вводе списка с помощью `input` пользователь может поместить в список данные любого типа. В приведенном выше примере после применения `eval()` получается список, состоящий из двух чисел и строки.

Чтобы убедиться, что элемент списка является строкой, метод `.isalpha()` уже применить нельзя. Числа таким методом не обладают и проверка закончится ошибкой.

Проблема определения принадлежности объекта данных, к определенному типу может быть решена с помощью встроенной функции-*предиката* **`isinstance()`**.

У этой функции два аргумента: *объект данных* и *тип данных*.

Например, для проверки принадлежности объекта данных `obj` к строковому типу нужно выполнить вызов **`isinstance(obj, str)`**, где **`str`** именует строковый тип данных. Если объект `obj` является строкой, то такой предикат возвратит `True`.

### Удаление элементов из списка в цикле

Для удаления элемента со значением `elem` из списка `lst` применяется метод списковых объектов `lst.`**`remove`**`(elem)`.

Этот метод находит и удаляет из списка *первый* встретившийся элемент, имеющий значение `elem`. Например,

```
>>> lst =['a','b','b','c']
>>> lst.remove('b')
>>> lst
['a','b','c']
```

Если поставить задачу удаления из списка всех элементов `'b'`, то можно воспользоваться циклом. Например,

```
for elem in lst:
    if elem=='b':
          lst.remove('b')
```

Однако после выполнения это кода будет получен тот же результат `['a','b','c']`, что и в предыдущем примере.

Причина простая: когда удаляется элемент с индексом `1` (первый из элементов `'b'`), то его место в списке занимает элемент, ранее имевший индекс `2` (т.е. второй элемент `'b'`). Но оператор цикла работает независимо. Он уже просматривал элемент с индексом `1`, поэтому переходит к следующему, т.е. к `'c'`.

Один из самых простых способов решить эту проблему — вместо удаления элементов из исходного списка создать *новый* пустой список и наоборот, последовательно добавлять в него те элементы, которые нужно сохранить.

```
>>> lst =['a','b','b','c']
>>> lst1=[]
>>> for elem in lst:
         if elem != 'b':
           lst1.append(elem)
>>> lst1
['a', 'c']
```

***Замечание***

Есть другие, технически немного более сложные приемы решения проблемы. Например, просматривать элементы списка с конца или после каждого удаления выходить из цикла и заново запускать его.

**ЗАДАНИЕ 3** (*Ввод слов списком*)

Дальнейшее расширение функциональности программы заключается в предоставлении пользователю возможности задавать слова, которые нужно проверить на принадлежность к палиндромам, не по одному, а сразу списком (*англ.* word list — список слов).

***1. Ввод списка слов.***

В функции **`get_word_list_1`**`()` выполняются следующие действия

а) Пользователю предлагается подсказка

"Введите список слов в формате [..., ..., ...]: ".

Введенная информация с помощью `eval` преобразуется из строки в список и запоминается в переменной `word_list`.

б) После получения списка `word_list` программа проверяет допустимость каждого элемента списка. А именно, нужно "отсечь"

- все данные, которые не являются строковыми (первый уровень контроля),
- из строковых элементов оставить только те, которые представляют слова из букв (второй уровень контроля).

Проблема определения, является ли введенный элемент информации строкой или нет, решается с помощью функции-*предиката* **`isinstance()`**.

Если встречается некоторый элемент `w`, который не является строкой, то программа сообщает

`"w — не является строкой"`

и должна удалить его из списка.

Из-за описанной выше проблемы с работой цикла `for-in`, вместо удаления элементов строится новый список, состоящий из допустимых элементов. При этом выполняется проверка второго уровня: в список добавляются только те строки, которые представляют слова.

Когда анализ всех элементов списка закончен, то дальнейшее поведение программы зависит от результата.

Если были удалены все элементы (список пустой), то программа выдает сообщение `"Проблема с исходными данными!"` и завершает работу.

Если список содержит хотя бы один элемент, то функция `get_word_list_1()` возвращает список проверенных слов.

***2. Тестирование функции.***

Проверяется правильность работы функции `get_word_list_1()`.

Для этого требуется подготовить вспомогательный код, который осуществляет вызов этой функция, а возвращаемое ею значение выводит на экран.

Проверить возвращаемый результат функции при вводе пользователем:

а) списка допустимых слов, например, `["первый","второй","третий"]`,

б) списка, включающего допустимые и недопустимые данные, например, `["первый", 2 ,"третий"]`,

в) список из недопустимых данных, например, `[1, 2, 3]`.

## Вопросы для самоконтроля

1. Что понимается под метапрограммированием?
2. Для чего предназначена функция `eval`?
3. Чем функция `eval` отличается от `exec`?
4. В чем отличия функции-предиката `isinstance()` от методов-предикатов `isalpha()`, `isnumeric()` и других?
5. В чем состоит проблема, возникающая при удалении из списка в цикле элементов с определенным значением?
6. Чем отличается работа операторов цикла с предусловием и постусловием?
7. Для чего в теле операторов цикла применяются инструкции `break` и `continue`?
8. Какую общую структуру имеет оператор `while` в Питоне?
9. Какую общую структуру имеет оператор `for-in` в Питоне?
10. Когда выполняются инструкции, помещенные в блок `else` операторов цикла?
11. Как с помощью `while` и `break` можно имитировать работу цикла с постусловием?

## *Справочная информация*

### Операторы цикла

В языках программирования операторы цикла предназначены для многократного выполнения подряд одного и того же набора инструкций. Каждое повторение называется *шагом цикла*.

Синтаксически операторы цикла строятся из *заголовка* и *тела*.

*Заголовок* определяет тип оператора цикла и содержит логическое выражение, вычисляемое заново на каждом шаге цикла и управляющее его работой. В зависимости от полученного результата происходит либо очередное выполнение цикла (`True`), либо оператор цикла завершает свою работу (`False`).

Тело цикла — это набор инструкций, которые выполняются на каждом шаге цикла.

Вычисление логического выражения и проверка его результата может происходить в разное время.

• Если проверка происходит до выполнения тела оператора, то имеем оператор цикла *с предусловием*. В этом случае, когда логическое выражение изначально имеет значение `False`, тело оператора цикла не будет выполнено ни разу.

• Если логическое выражение вычисляется после выполнения инструкций тела цикла, то имеем оператор цикла *с постусловием*. В этом случае тело оператора цикла будет выполнено, как минимум, один раз.

**а) *Оператор цикла* `while`.**

В Питоне имеется оператор цикла с предусловием while, который в простой форме имеет вид

```
while логическое_выражение:
        инструкции (тело цикла)
```

Для управления выполнением оператора while в его теле могут использовать две специальных инструкции:

break — заверить работу оператора, независимо от значения логического выражения (досрочное прекращение циклических повторений),

continue — перейти к следующему шагу, независимо от того, завершено ли выполнение тела цикла на текущем шаге, например, когда нужно пропустить остаток текущего шага.

В более сложной форме оператор `while` имеет дополнительную конструкцию `else`, содержащую свой блок инструкций. Эти инструкции выполняются в том случае, когда работа цикла завершится естественным образом, и не выполняется, если выход из цикла произошел в результате выполнения оператора `break`. Общая структура оператора:

```
while логическое_выражение:
   инструкции (тело цикла)
else
    дополнительные инструкции
```

Несмотря на то, что в Питоне нет оператора цикла с постусловием, такой тип циклов в нём можно смоделировать с помощью связки `while-break`.

```
while True:
  инструкции
  if логическое_выражение:    # условие выхода из цикла
       break;
```

**б) *Оператор цикла* `for-in`.**

В современных языках программирования имеется другой тип операторов цикла, предназначенных для обработки *итерируемых* объектов, т.е. объектов, для которых определен способ извлечения составляющих их элементов по одному. Так как последовательности (строки, списки, кортежи) позволяют обращаться к отдельным элементам по индексу, то они относятся к итерируемым объектам.

В Питоне для работы с итерируемыми объектами целей предлагается оператор `for-in`. В его заголовке указывается итерируемый объект и имя переменной, в которой на каждом шаге будут запоминаться извлекаемые из коллекции элементы. Цикл прекращает свою работу, когда в итерируемом объекте исчерпаны все элементы.

Как и в случае `while`, в теле цикла могут использоваться операторы `break` и `continue`. Кроме того, может присутствовать конструкция `else` с набором инструкций, которые выполняются при нормальном завершении циклических повторений и не выполняются, если выход из цикла был вызван выполнением `break`.

```
for переменная in итерируемый_объект:
   инструкции (тело цикла)
else
    дополнительные инструкции
```

По характеру работы оператор `for-in` можно отнести к циклам с предусловием. Например, следующий фрагмент кода ничего не выведет на экран, так как предлагаемый список пуст:

```
>>> for x in []
        print(1)
```