8 РАБОТА СО СТРОКАМИ

8.1 Основные понятия

Следует отметить, что первоначальные навыки работы со строками в языке программирования Python мы получили ранее. Рассмотрено как сделать приглашение к вводу данных или вывести ответ в программе, также такие вопросы как осуществление перевода строки, табулирование, форматирование строки. В этой теме познакомимся с функциями и методами, предназначенными для работы со строками.

Строка в Python - это объект класса **str**. В темах ранее, для того чтобы работать с числовыми данными, применялись функции приведения типов, например, **int**, как это продемонстрировано в следующем примере:

```
ind=int(input(" \n Введите индекс элемента "))
```

На рисунке показано, что строки индексируются с нуля, то есть, если имеется оператор **stroka="python"**, то первый символ в строке нулевой и обращение к нему будут записываться как **stroka[0]**. В то же время к элементам строки в Python может обращаться, указывая отрицательные индексы, например, оператор **print(stroka[-6])** есть ничто иное, как вывод символа **p** на экран.

0	1	2	3	4	5
p	y	t	h	o	n
-6	-5	-4	-3	-2	-1

Рисунок 103 – Индексация строки

Известно, что в программировании, прежде чем сравнить один символ с другим, его следует преобразовать в число с помощью таблицы ASCII (American Standard Code for Information Interchange - Американский стандартный код для обмена информацией), который поддерживает кодирование 128 буквенно-цифровых символов.

Первые 32 кода базовой таблицы, начиная с нулевого, отданы разработчикам аппаратных средств (в первую очередь производителям компьютеров и печатающих устройств). В этой области размещаются так называемые управляющие коды, которым не соответствуют никакие символы языков, и соответственно, эти коды не выводятся ни на экран, ни на

устройства печати, но используются для функций управления, которые мы приведем в таблице.

Таблица 7 - Первые 32 кода базовой таблицы, начиная с нулевого

Символ	Код	Клавиши	Значение
nul	0	Ctrl + @	Нуль
soh	1	Ctrl + A	Начало заголовка
stx	2	Ctrl + B	Начало текста
etx	3	Ctrl + C	Конец текста
eot	4	Ctrl + D	Конец передачи
enq	5	Ctrl + E	Запрос
ack	6	Ctrl + F	Подтверждение
bel	7	Ctrl + G	Сигнал (звонок)
bs	8	Ctrl + H	Забой (шаг назад)
ht	9	Ctrl + I	Горизонтальная табуляция
lf	10	Ctrl + J	Перевод строки
vt	11	Ctrl + K	Вертикальная табуляция
ff	12	Ctrl + L	Новая страница
cr	13	Ctrl + M	Возврат каретки
so	14	Ctrl + N	Выключить сдвиг
si	15	Ctrl + O	Включить сдвиг
dle	16	Ctrl + P	Ключ связи данных
dc1	17	Ctrl + Q	Управление устройством 1
dc2	18	Ctrl + R	Управление устройством 2
dc3	19	Ctrl + S	Управление устройством 3
dc4	20	Ctrl + T	Управление устройством 4
nak	21	Ctrl + U	Отрицательное подтверждение
syn	22	Ctrl + V	Синхронизация
etb	23	Ctrl + W	Конец передаваемого блока
can	24	Ctrl + X	Отказ
em	25	Ctrl + Y	Конец среды
sub	26	Ctrl + Z	Замена
esc	27	Ctrl + [	Ключ
fs	28	Ctrl + \	Разделитель файлов
gs	29	Ctrl +]	Разделитель группы
rs	30	Ctrl + ^	Разделитель записей
us	31	Ctrl + _	Разделитель модулей

Национальные стандарты кодировочных таблиц включают международную часть кодовой таблицы без изменений, а во второй половине содержат коды национальных алфавитов, символы псевдографики и некоторые математические знаки.

Начиная с кода 32 по код 127 размещены коды символов английского алфавита, знаков препинания, цифр, арифметических действий и некоторых вспомогательных символов. Базовая таблица кодировки ASCII приведена в таблице 8.

Таблица 8 - Базовая таблица кодировки ASCII

sp 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	

Таким образом, при работе со строками важно знать, что символ "**A**" - это совсем не то же самое, что символ "**a**". Длина строки ограничена лишь объемом оперативной памяти компьютера, и для обращения к элементу строки достаточно указать его индекс в квадратных скобках. Подобным образом ранее рассматривалось обращение к элементу списка.

Строки в Python представляют собой неизменяемую последовательность и обрабатываются с использованием цикла с оператором **for**. Важно понять, что изменить символ строки, обратившись к нему по индексу, невозможно.

```
stroka="Python"
stroka[0]="p"  #недопустимый оператор
```

При обработке строки к ней можно обратиться непосредственно в цикле, как это показано в листинге ниже.

```
stroka="python"
for i in stroka:
    print(i, end=" ")
```

Результатом работы этого кода станет вывод на экран слова **p y t h o n**.

8.2 Функции для работы с символами

Рассмотрим основные функции, предназначенные для работы со строками.

1. ***Функция len().***

При работе со строками может оказаться полезной функция **len()**, назначение которой заключается в определении длины строки. В операторе **print** мы обращаемся к каждому элементу строки, упоминая имя строки и заключая индекс каждого элемента в квадратные скобки. Естественно, результатом нижеследующего кода станет вывод на экран слова **P y t h o n**.

```
stroka="Python"
for i in range(len(stroka)):
   print(stroka[i], end=" ")
```

2. Функция ord().

Другая функция - **ord()**, синтаксис которой **ord("Символ")** возвращает код указанного символа. Например, функция **ord()**, которую мы применили к каждому символу строки в нижеследующем примере, возвратит ASCII-код символов строки "**Python**".

```
stroka="Python"
for i in range(len(stroka)):
   print(ord(stroka[i]), end=" ")
```

Результат работы программы представлен на рисунке 104.

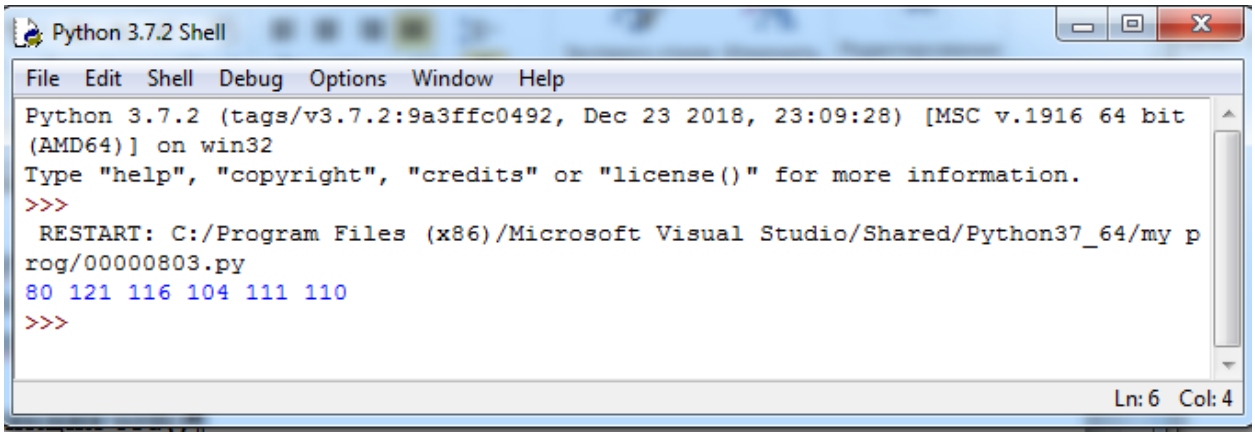

Рисунок 104 – Выведены ASCII-коды символов слова Python

3. Функция chr().

Действие функции, синтаксис которой **chr(Число)**, прямо противоположно функции **ord()**, а именно, по ASCII-коду символа вернуть символ. Таким образом, функция **chr()**, которую применили к функции **ord()**, которая, в свою очередь, возвращала ASCII-коды символов, вернет исходную строку "Python" в нижеприведенном коде.

```
stroka="Python"
for i in range(len(stroka)):
   print(chr(ord(stroka[i])), end=" ")
```

8.3 Методы работы со строками

Рассмотрим основные методы, предназначенные для работы со строками.

1. Метод upper().

Синтаксис метода: **stroka.upper()**. Преобразует все символы строки к верхнему регистру. Например,

```
stroka="PythOn"
newstr=stroka.upper()
print(newstr)
```

2. Метод lower().

Синтаксис метода: **stroka.lower()**. Преобразует все символы строки к нижнему регистру. Например,

```
stroka="PytHon"
newstr=stroka.lower()
print(newstr)
```

3. Метод swapcase().

Синтаксис метода: **stroka. swapcase()**. Преобразует все символы строки, записанные в нижнем регистре - в верхний и наоборот как показано на рисунке 105. Например,

```
stroka="PytHoN"
newstr=stroka.swapcase()
print(newstr)
```

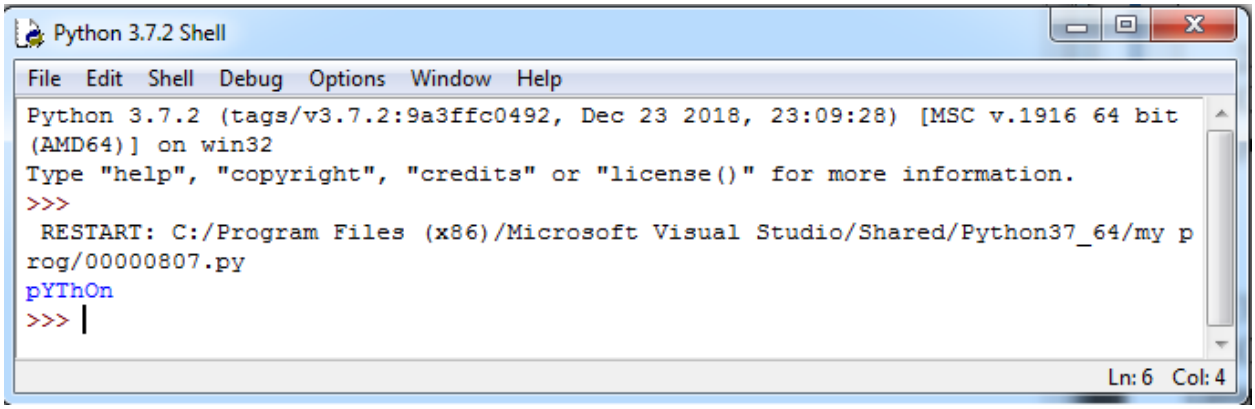

Рисунок 105 – Работа метода swapcase()

4. Метод capitalize().

Синтаксис метода: **stroka.capitalize()**. Преобразует первую букву в строке в верхний регистр, а остальные в нижний регистр. Например,

```
stroka="pyTHoN"
newstr=stroka.capitalize()
print(newstr)
```

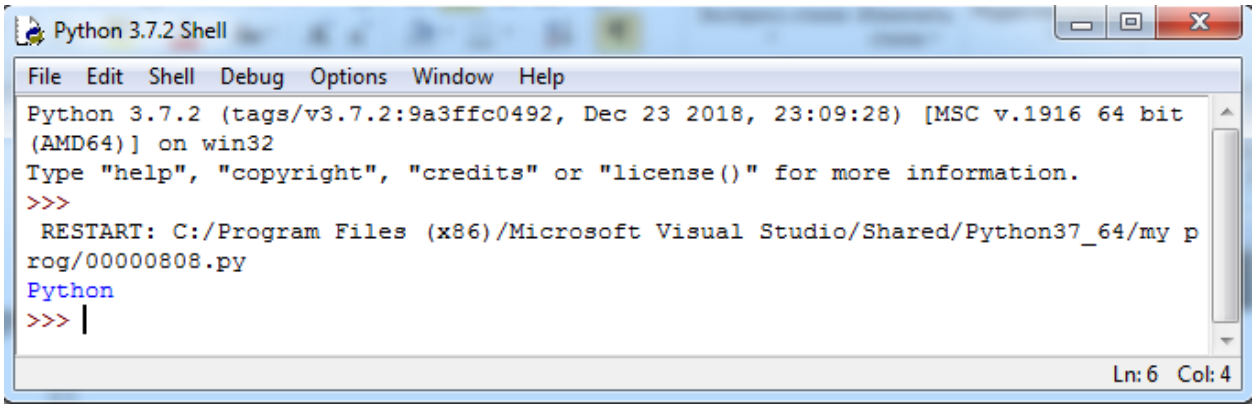

Рисунок 106 – Работа метода capitalize()

5. Метод title().

Синтаксис метода: **stroka.title()**. Преобразует, все первые буквы в строке в верхний регистр, а остальные в нижний регистр как показано на рисунке 107. Например,

```
stroka="complete washingtoN quarTers national paRks"
newstr=stroka.title()
print(newstr)
```

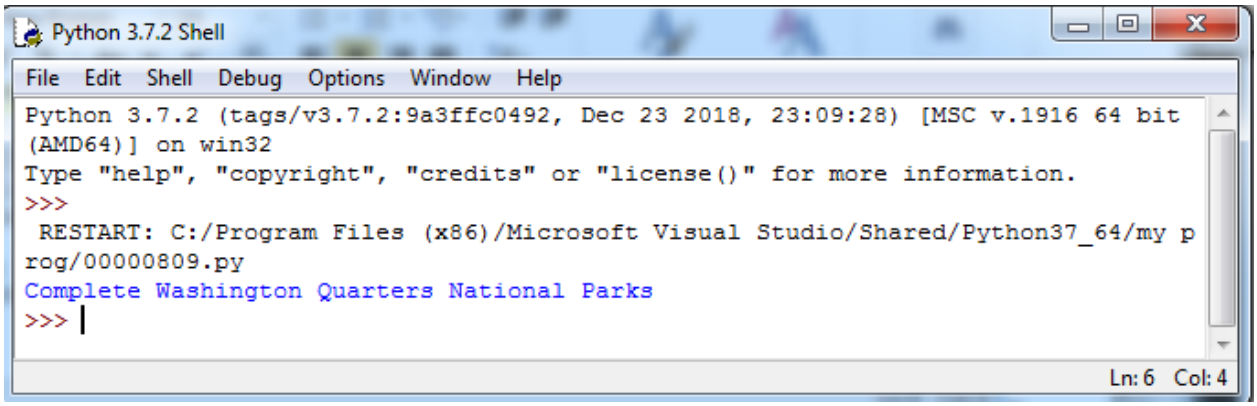

Рисунок 107 – Работа метода title()

6. Метод startswith().

Синтаксис метода: **stroka.startswith(podstroka)**. Проверяет, начинается ли строка **stroka** с указанной подстроки (фрагмента строки) **podstroka**. Например, в данном примере проверяется условие наличия подстроки "compl" в начале строки "complete washingtoN quarTers national paRks". Результатом данного кода будет значение **True**.

```
stroka="complete washingtoN quarTers national paRks"
podstr="compl"
```

```
n=stroka.startswith(podstr)
print(n)
```

7. Метод endswith().

Синтаксис метода: **stroka.endswith(podstroka)**. Проверяет, заканчивается ли строка **stroka** указанной подстрокой **podstroka**. Например, в данном примере проверяется условие наличия подстроки "rks" в конце строки "complete washingtoN quarTers national paRks". Результатом данного кода будет значение **False.** Это еще раз говорит о чувствительности ввода текста к регистру, так как наша строка заканчивается как "Rks", а не "rks".

```
stroka="complete washingtoN quarTers national paRks"
podstr="rks"
n=stroka.endswith(podstr)
print(n)
```

8. Метод replace().

Синтаксис метода: **stroka.replace(old, new)**, где **old** - подстрока для замены, **new** - новая подстрока. Метод находит и заменяет в строке **stroka** подстроку **old** подстрокой **new**. Например, в данном примере оператор stroka=stroka.replace("washingtoN quarTers national paRks ","Washington quarters National parks") возвратит строку "Complete Washington quarters National parks".

```
stroka="Complete washingtoN quarTers national paRks"
stroka=stroka.replace("washingtoN quarTers national paRks", "Washington quarters National parks")
print(stroka)
```

9. Метод rfind().

Синтаксис метода: **stroka.rfind(podstr)**, где **podstr** - подстрока. Метод возвращает позицию последнего вхождения подстроки в строку. Если **подстрока не обнаружена**, то возвращается значение **-1**. После параметра **podstr** могут быть указаны начальная позиция и конечная позиции в строке, где следует искать подстроку. Эти параметры необязательны, и если отсутствует начальная позиция, то поиск будет осуществлен с начала строки. Так, в нижеприведенном примере, последнее **вхождение подстроки "on"** будет зафиксировано **на позиции 17**, так как мы указали в поиске в качестве **начальной позиции 8** символ, а **конечной 30**. Если же в примере **не указывать начальную и конечную позиции**, то результатом будет **33**. Строка, как мы помним, индексируется с нуля, пробелы и знаки препинания являются также символами.

```
stroka="Complete Washington quarters National parks"
stroka=stroka.rfind("on", 8, 30)
```

```
print(stroka)
```

10. Метод find().

Синтаксис метода: **stroka.flnd(podstr)**, где **podstr** - подстрока. В отличие от предыдущего метода, метод **find()** возвращает позицию первого вхождения подстроки в строку. Если **подстрока не обнаружена**, то возвращается значение **-1**. Соответственно, если изменить предыдущую программу и вместо метода **rfind()** использовать метод **find()**, а также убрать в поиске начальную и конечную позиции, то первое вхождение подстроки "on" в строку "Complete Washington quarters National parks" будет начинаться на позиции 17.

```
stroka="Complete Washington quarters National parks"
stroka=stroka.find("on")
print(stroka)
```

11. Метод count().

Синтаксис метода: **stroka.count(podstr)**, где **podstr** - подстрока. Метод возвращает количество вхождений подстроки **podstr** в строку **stroka**. Таким образом, в нижеследующем примере ответом, выведенным на экран оператором **print**, будет число **2**, поскольку подстрока "on" входит два раза в строку "Complete Washington quarters National parks".

```
stroka="Complete Washington quarters National parks"
stroka=stroka.count("on")
print(stroka)
```

12. Метод strip().

Синтаксис метода: **stroka.strip()**. Метод удаляет начальные и конечные пробелы в строке **stroka**. В нижеприведенном примере в исходной строке присутствуют пробелы в начале и в конце строки.

```
stroka="      Complete Washington quarters National parks     "
stroka=stroka.strip()
print(stroka)
```

На рисунке 108 представлен результат выполнения программы, где пробелы с помощью метода **strip()** удалены из исходной строки.

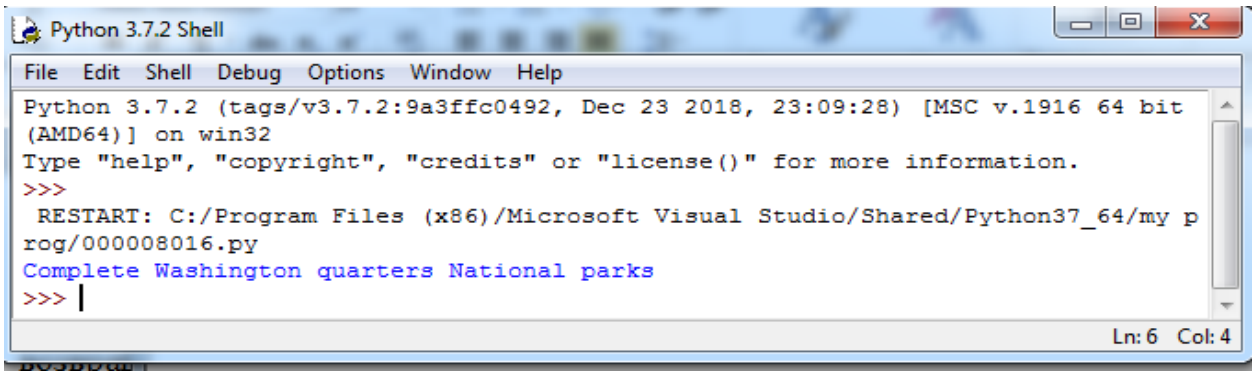

Рисунок 108 – Работа метода strip()

13.Методы lstrip() и rstrip().

Их синтаксис и действие аналогичны рассмотренному методу **strip()**, с той разницей, что метод **lstrip()** удаляет символы пробела слева от начала исходной строки, а метод **rstrip()** - в конце исходной строки.

Отметим, что с помощью методов **strip()**, **rstrip()** и **lstrip()** можно удалять не только пробелы. Так, если в качестве параметра одного из методов указать символ или последовательность символов, то произойдет его (их) удаление. Например, в нижеследующем примере мы указали в качестве параметра в методе **strip()** символы "Complete ". Соответственно, метод возвратит исходную строку, но уже без указанных символов.

```
stroka="Complete Washington quarters National parks     "
stroka=stroka.strip("Complete ")
print(stroka)
```

Подобная ситуация показана на рисунке 109.

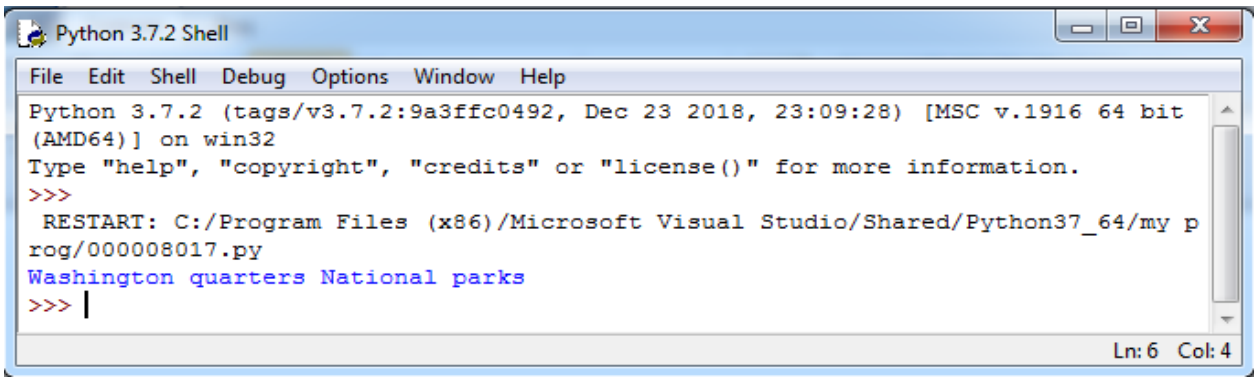

Рисунок 108 – Работа метода strip() с параметром

14. Метод split().

Синтаксис метода: stroka. **split().** Метод разделяет строку на подстроки и добавляет их в список. Если в качестве параметра присутствует разделитель, то строка будет разбита на подстроки в соответствии с указанным разделителем. В случае его отсутствия разделителем считается пробел. В коде, представленном в листинге ниже, исходная строка методом **split()** преобразована в список, разделителем служит пробел. Далее уже над

списком выполняется списочный метод **remove()**, который удаляет тот элемент списка, который вводит пользователь с клавиатуры.

```
stroka="America the Beautiful quarters: Complete Washington quarters National parks"
spisok=stroka.split(" ")
print(spisok)
element=input("\n Введите элемент для удаления его из списка: ")
for i in spisok:
   if element in spisok:
      spisok.remove(element)
print("\n",spisok)
```

Результат работы программы представлен на рисунке 109. Следует обратить внимание на то, что при формировании списка разделителем был пробел. Соответственно, в исходном списке элемент списка **"quarters"** входит в него один раза, также как и элемент списка **"quarters:"**. При запросе удаляемого элемента, мы ввели **"quarters"**, поэтому элемент списка **"quarters:"** остался в списке, а удален из списка только **"quarters"**.

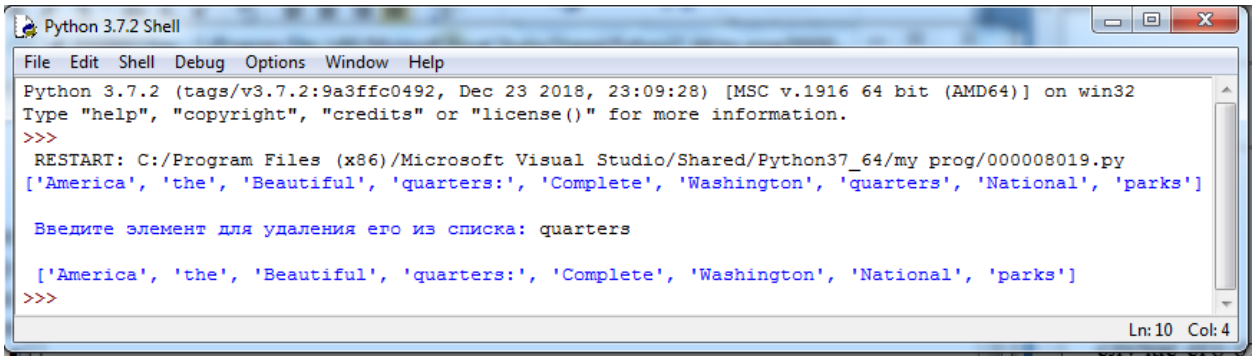

Рисунок 109 – Работа метода split() со списочным методом remove() для удаления из списка элемента "quarters"

15. Метод join().

Синтаксис метода: **разделитель.join(spisok)**, где **spisok** - список или кортеж. Еще одним способом преобразования строки в список является использование функции **list()**, а обратное преобразование осуществляется методом]**join()**. Такие преобразования необходимы для того, чтобы изменить элемент строки по индексу, поскольку изначально строки неизменяемы.

В нижеприведенном примере из исходной строки с помощью функции **list()** происходит формирование списка. Попробуем заменить элемент исходного списка на введенный с клавиатуры и для этого осуществляем соответствующие запросы с помощью функции **input()**. Вычисляем длину списка с помощью функции **len()** и организуем просмотр списка с помощью оператора цикла **for**. Если очередной элемент списка будет равен введенному элементу, то произойдет замена исходного элемента списка **element** на

элемент **element1**. Затем, используя метод **join()**, преобразуем список в строку и выводим ее на экран (рисунок 110).

```
stroka="America the Beautiful quarters: Complete Washington quarters National parks"
spisok=list(stroka)
print(spisok)
element=input("\n Введите элемент для замены его в списке: ")
element1=input("\n Введите элемент, на который нужно поменять: ")
n=len(spisok)
for i in range(0, n):
    if spisok[i]==element:
        spisok[i]=element1
stroka1="".join(spisok)
print("\n",stroka1)
```

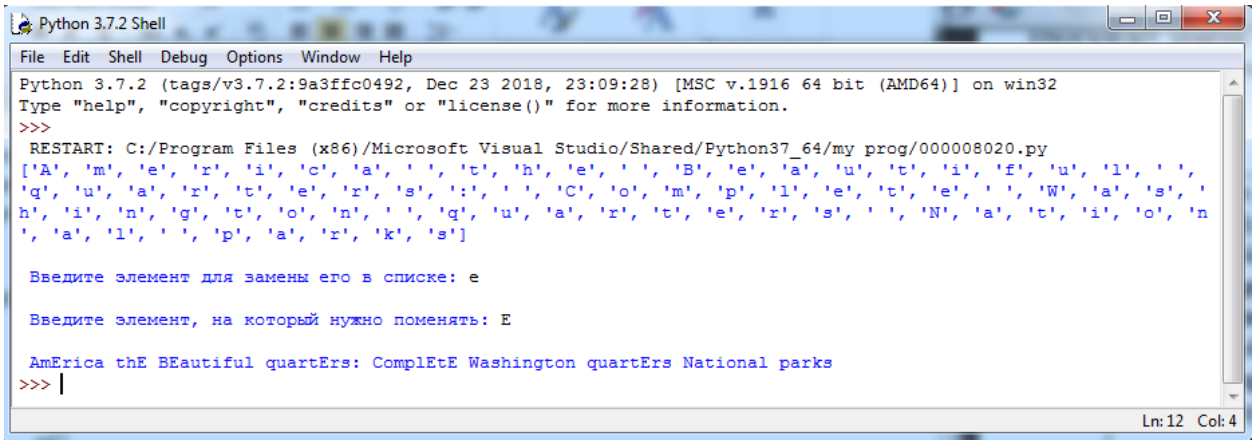

Рисунок 110 – Замена элемента списка "e" на "E"

8.4 Базовые алгоритмы обработки строк

Вышеприведенные примеры с использованием функций и методов дают некоторое представление о возможностях обработки строк и символов в Python. При решении задач следует научиться использовать в совокупности простые приемы и методы обработки строк (назовем их базовыми алгоритмами), среди которых можно выделить следующие:

1. Определение количества символов.
2. Замена символов в строке.
3. Удаление символа в строке.
4. Вставка символа в строку.
5. Анализ символа на принадлежность к группе.
6. Обращение строки.
7. Алфавитная выборка.
8. Срезы строк.

Данные базовые алгоритмы могут найти свое применение не только для решения учебных задач, но и для разработки простых текстовых

редакторов, программного обеспечения предназначенного для верстки текста и т. д.

Определение количества символов.

Задача 8.4.1. Пользователь вводит исходную строку **stroka** и символ **simvol**. Подсчитайте, сколько раз заданный символ встречается в исходной строке.

Решение. Первоначально после обнуления ячейки **k**, которая будет играть роль счетчика символов, мы преобразуем все символы исходной строки в строчные буквы методом **lower()**. Затем в цикле происходит сравнение текущего символа строки **spisok[i]** с искомым символом **simvol**. Параметр цикла **i** будет изменяться от **0** (начало строки) до конца строки (за это отвечает ячейка **n**, в которой уже находится значение функции **len(spisok)**). Подсчет количества найденных символов в строке обеспечивает оператор **k+=1**.

Ниже приведен код программы, отвечающий за решение задачи.

```
k=0
stroka=input("\n Введите строку: ")
stroka1=stroka.lower()
spisok=list(stroka1)
simvol=input("\n Введите символ для поиска его повторений в списке: ")
n=len(spisok)
for i in range(0, n):
    if spisok[i]==simvol:
        k+=1
print("\n", k)
```

Результат работы программы представлен на рисунке 111.

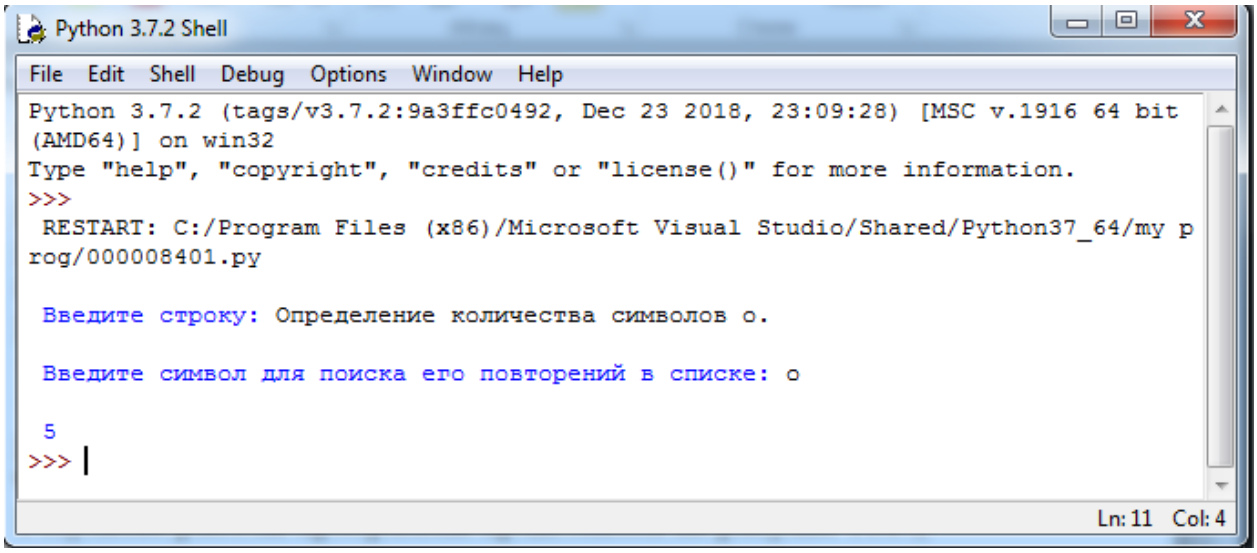

Рисунок 111 – Результат работы программы по определению количества символов в строке

Замена символов в строке.

Задача 8.4.2. Пользователь вводит исходную строку **stroka** и символ **simvol**. Замените пробелы в исходной строке указанным символом.

Решение. Ранее при объяснении работы метода **join()** была решена подобная задача. В листинге приведен код, в котором обрабатываемая строка не фиксирована, а вводится с клавиатуры. Методы ее обработки остались прежними. Заметим, что подобную задачу можно решить и с применением метода **replace().**

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
stroka1=stroka.lower()
spisok=list(stroka1)
simvol=input("\n Введите символ для замены им пробелов: ")
n=len(stroka1)
for i in range(0, n):
    if spisok[i]==" ":
        spisok[i]=simvol
stroka1="".join(spisok)
print("\n", stroka1)
```

Результат работы программы представлен на рисунке 112.

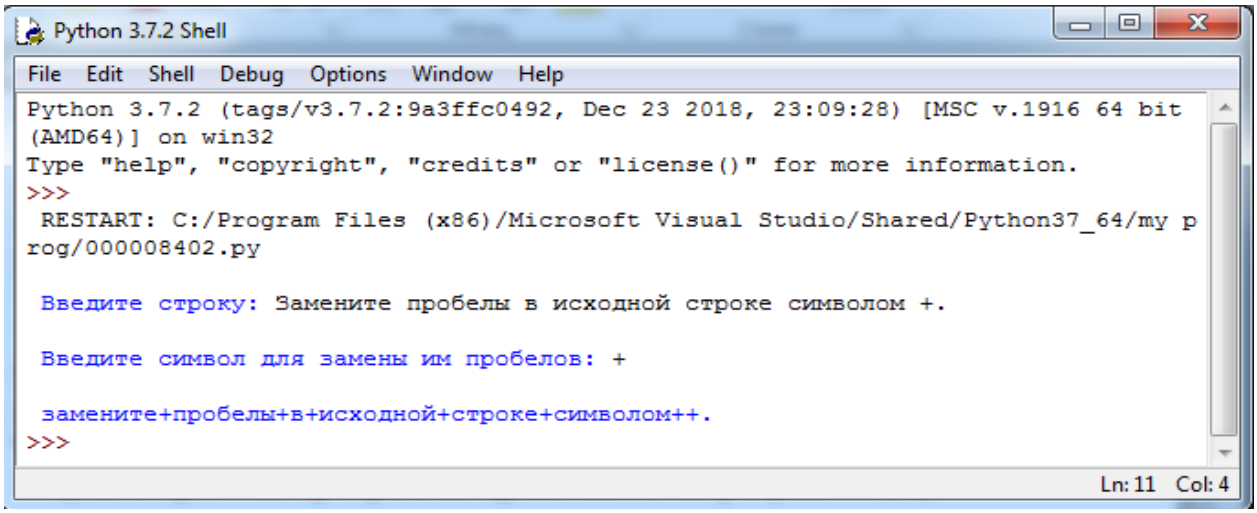

Рисунок 112 – Результат работы программы по замене пробелов в исходной строке символом +

Удаление символов в строке.

Задача 8.4.3. Пользователь вводит исходную строку **stroka** и символ **simvol**. Удалите в исходной строке указанный символ.

Решение. Способ основан на применении метода **replace()**. Осуществляем ввод символа, который хотим удалить, а затем используем его

в качестве параметра в методе **replace()**. Второй параметр в методе делаем равным пустым кавычкам.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
simvol=input("\n Введите символ для удаления его из строки: ")
stroka=stroka.replace(simvol, "")
print("\n", stroka)
```

Результат работы программы представлен на рисунке 113.

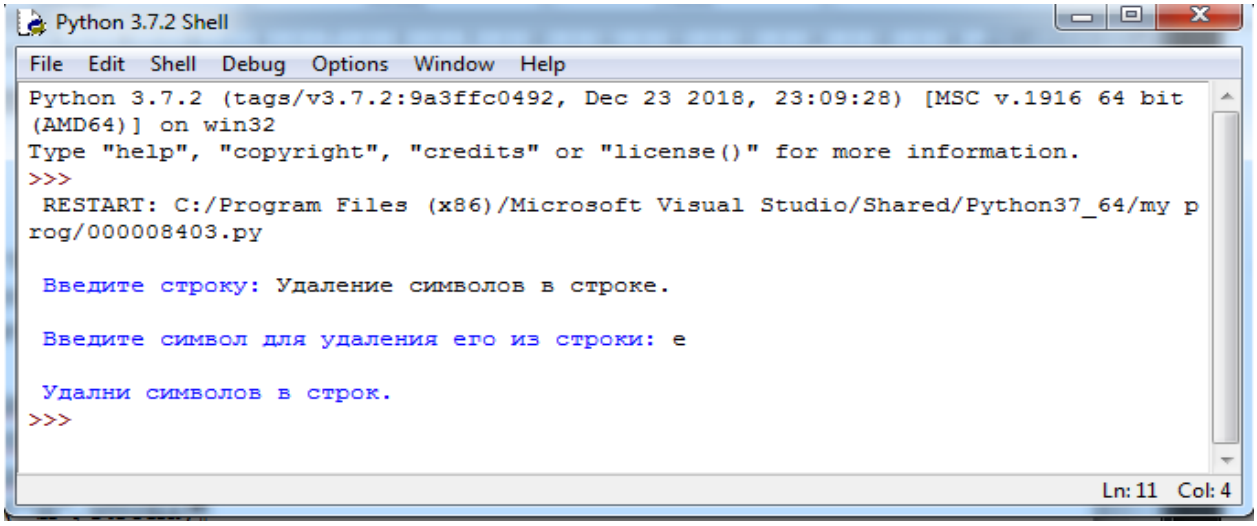

Рисунок 113 – Результат работы программы по удалению введенного символа в исходной строке

Вставка символа в строку.

Задача 8.4.4. Пользователь вводит исходную строку **stroka.** В исходную строку необходимо добавить символ **simvol1** после символа **simvol**, которые пользователь вводит с клавиатуры.

Решение. Вставка подстроки в заданную позицию осуществляется с помощью оператора **spisok.insert(i+l, simvol1)**, предварительно мы преобразовали строку в список с помощью функции **list**. Изменение параметра **i** в цикле с оператором **for** обеспечивает продвижение по строке. Если очередной элемент списка равен найденному элементу (**if spisok[i]== simvol**), то применяется метод **insert**. В конце программы, используя метод **join**, делаем обратное преобразование: список превращается в строку, которая и выводится на экран. Важно помнить, что при работе со строками надо учитывать, в каком состоянии находится символ - в верхнем или нижнем регистре.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
print(stroka)
```

```
spisok=list(stroka)
simvol=input("\n Введите символ, после которого вставим введенный ниже символ: ")
simvol1=input("\n Введите символ для вставки его в строку: ")
n=len(stroka)
k=0
for i in range(0, n):
    if spisok[i]==simvol:
        k+=1
for i in range(0, n+k):
    if spisok[i]==simvol:
        spisok.insert(i+1, simvol1)
stroka="".join(spisok)
print("\n", stroka)
```

Результат работы программы представлен на рисунке 114.

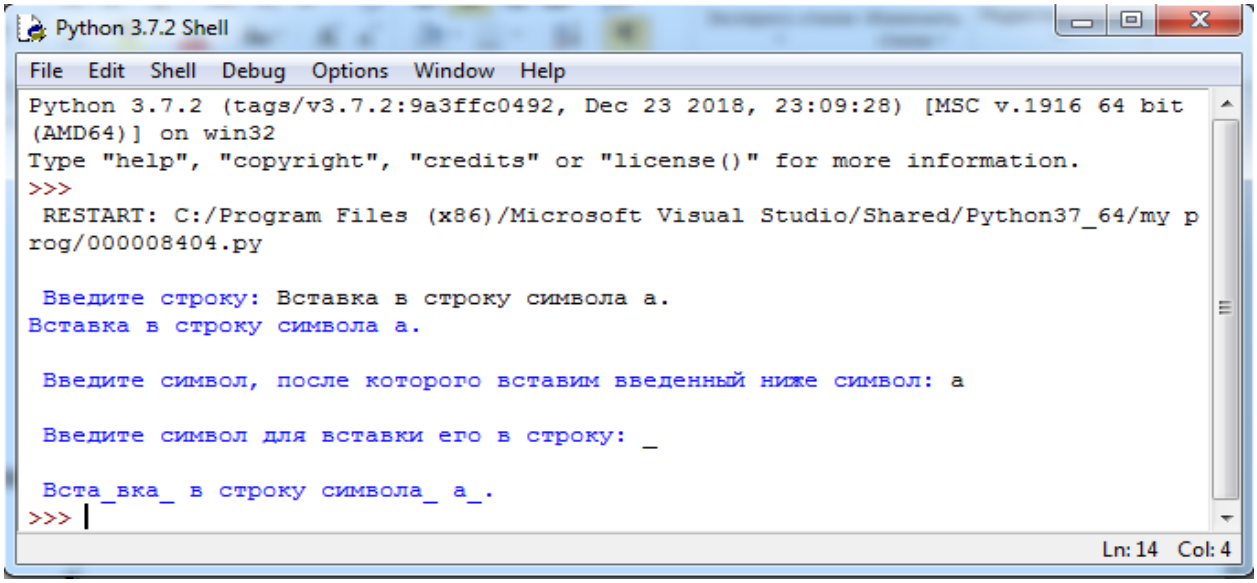

Рисунок 114 – Результат работы программы по вставке введенного символа "_" после символа "а" в исходную строку

Анализ символа на принадлежность к группе.

Задача 8.4.5. Пользователь вводит исходную строку **stroka.** Определить количество символов **simvol** и **simvol1** в исходной строке. Символы **simvol1** и **simvol** пользователь вводит с клавиатуры.

Решение. В данной программе, преобразовав строку символов в список, вводим два символа, количество которых мы хотим обнаружить в исходной строке. В цикле с оператором **for** сравниваем каждый элемент списка с введенными пользователем символами, и если условие истинно, счетчик увеличивается на единицу.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
print(stroka)
spisok=list(stroka)
simvol=input("\n Введите первый символ, количество которого необходимо найти: ")
simvol1=input("\n Введите второй символ, количество которого необходимо найти: ")
k1=0
k2=0
k=0
n=len(stroka)
for i in range(0, n):
    if spisok[i]==simvol:
        k+=1
        k1+=1
    if spisok[i]==simvol1:
        k+=1
        k2+=1
print("\n Всего найденных символов", simvol," = ", k1)
print("\n Всего найденных символов", simvol1," = ", k2)
print("\n Всего найденных символов", simvol, " и ", simvol1, " = ", k)
```

Результат работы программы представлен на рисунке 115.

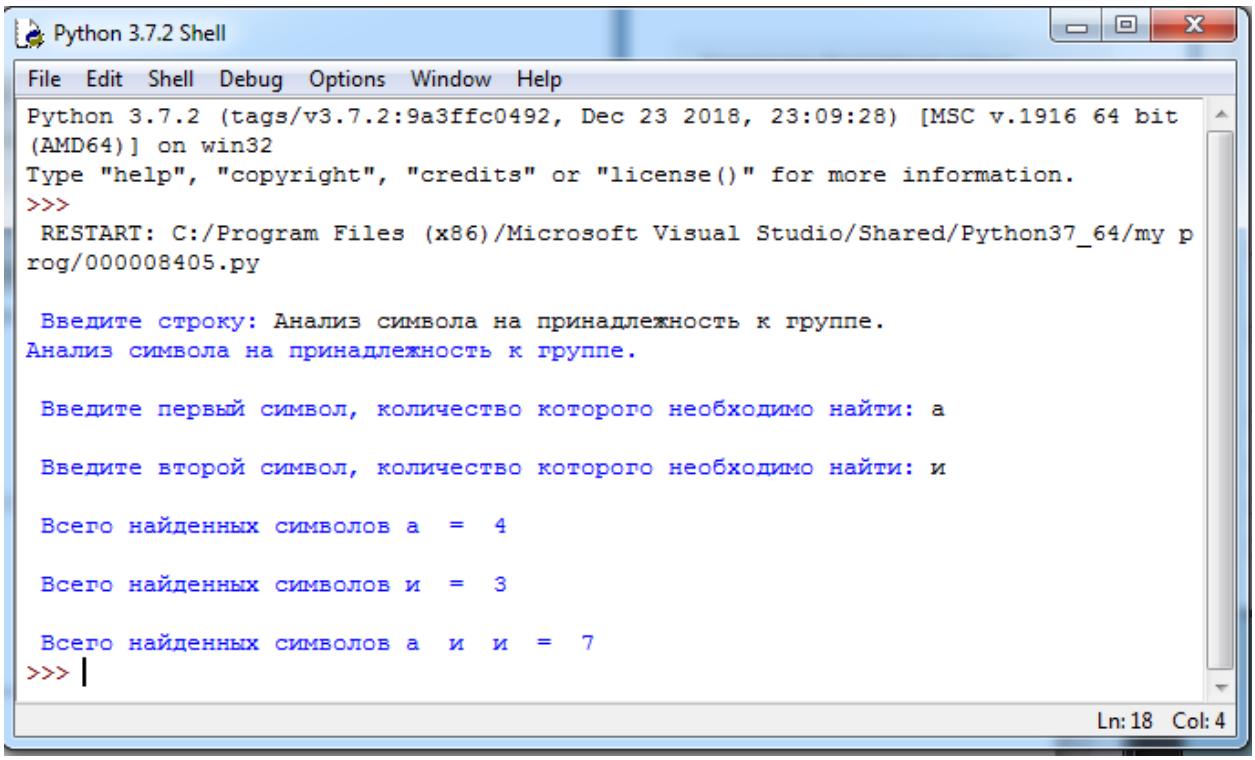

Рисунок 115 – Результат работы программы по анализу символов на принадлежность к группе

Обращение строки.

Задача 8.4.6. Пользователь вводит исходную строку **stroka.** Необходимо перевернуть строку, то есть записать символы в обратном порядке (последние становятся первыми и наоборот).

Решение. При решении данной задачи можно было бы воспользоваться методом **reverse** для обработки списков, но алгоритм работы данной программы основан на поочередной выборке символов из заданной строки и перемещение их в начало новой строки. Вспомогательная строка **tmp** изначально пуста. С помощью организации цикла просматриваем символы исходной строки от первого к последнему. Каждый из них присоединяется к началу уже накопленной строки, оператором **tmp=stroka[i]+tmp**.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
print("\n", stroka)
tmp=""
n=len(stroka)
for i in range(0, n):
    tmp=stroka[i]+tmp
print("\n Строка наоборот: ", tmp)
```

Результат работы программы представлен на рисунке 116.

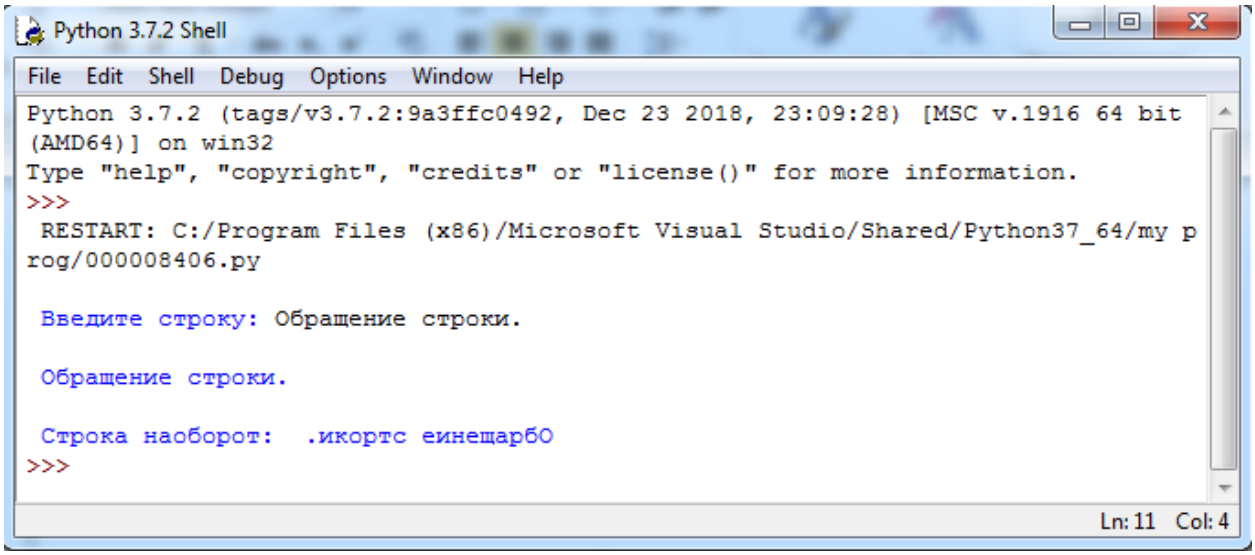

Рисунок 116 – Результат работы программы по обращение строки

Алфавитная выборка.

Задача 8.4.7. Имеется заранее заданный алфавит. Пользователь вводит исходную строку **stroka.** Выберите из строки символы, используемые в алфавите, и выведите их на экран.

Решение. При решении данной задачи мы познакомимся с таким понятием, как «константа» в Python. Переменная, набранная прописными буквами, в изучаемом языке программирования называется **константой**. Существует особенность применения констант в Python: в отличие от многих языков программирования константы можно изменять. Поэтому, создав константу, пользователь должен контролировать ее неизменность.

Итак, имеем некий набор символов, условно называемый алфавитом, который хранится в константе **ALFAVIT**. Пользователь вводит строку, и для каждого символа строки (**for** letter **in** stroka:) мы осуществляем проверку условия. Если символ есть в алфавите (**if** letter **in** ALFAVIT:), то в переменной **tmp** накапливаем результирующие символы оператором **tmp+=letter**.

Ниже приведен код программы, отвечающий за решение задачи.

```
ALFAVIT="abcdgh"
stroka=input("\n Введите строку: ")
print("\n", stroka)
tmp=""
for letter in stroka:
    if letter in ALFAVIT:
        tmp+=letter
print(tmp)
```

Результат работы программы представлен на рисунке 117.

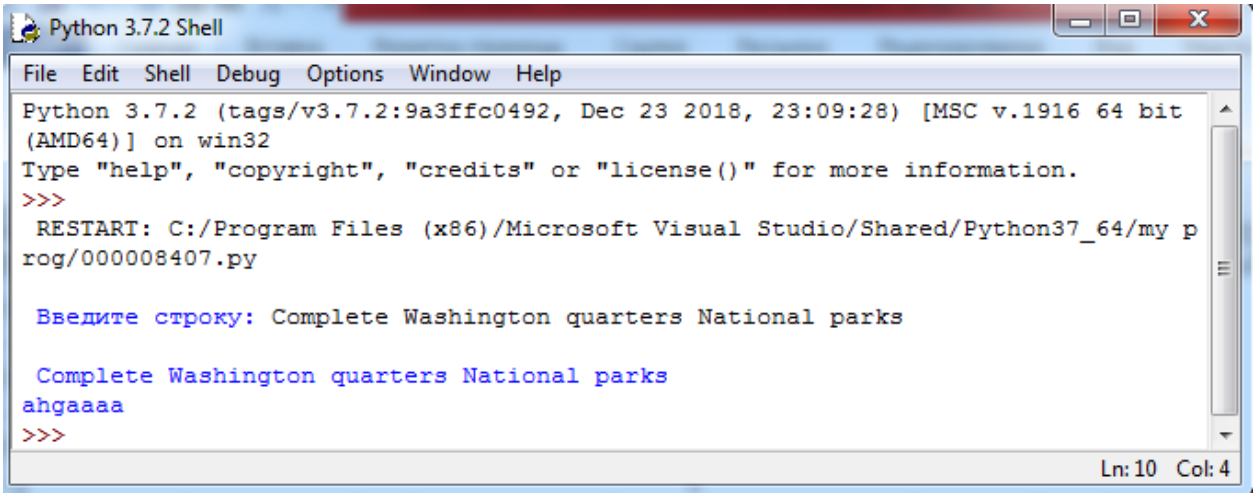

Рисунок 117 – Результат работы программы алфавитной выборки

Срезы строк.

Задача 8.4.8. Из исходной строки получите символы, расположенные между заданными начальным и конечным значениями.

Решение. При изучении темы по работе с кортежами, уже было знакомство с понятием «срез» и говорилось о том, что срез кортежа получается в результате вывода элементов кортежа, находящихся между

заранее заданными начальной (**a**) и конечной (**b**) позициями элементов. Механизм работы со срезами строк очень похож на принципы работы со срезами кортежей. Применяя срезы к строкам, мы можем выбрать из них любой символ или последовательность расположенных подряд символов. Для этого нам понадобятся начальная и конечная позиции, которые будут обозначать границы среза.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
print("\n", stroka)
flag=None
while flag!="":
   flag=input("\n Введите начало среза или для выхода введите ENTER: ")
   if flag:
      flag=int(flag)
      kon=input("\n Введите конец среза: ")
      kon=int(kon)
      print(stroka[flag:kon])
input("\n Введите ENTER для выхода ")
```

Результат работы программы представлен на рисунке 118.

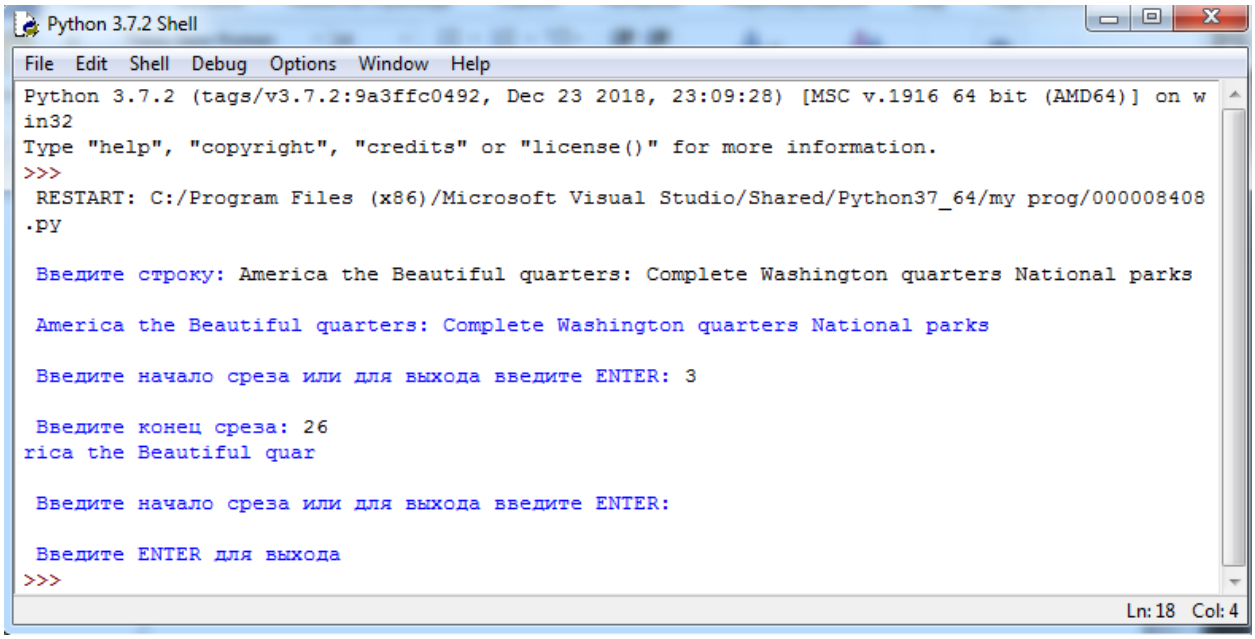

Рисунок 118 – Результат работы программы среза строки

8.5 Примеры решения задач

Задача 8.5.1. Пользователь вводит строку **stroka**, содержащую несколько слов, между которыми один или несколько пробелов. Требуется найти количество символов в самом длинном слове.

Решение. В цикле с оператором **for** мы проверяем наличие пробела в строке, указывающее на переход к очередному слову. В переменной **kol** будет храниться количество символов в слове. Соответственно, для того чтобы найти максимальную длину слова, мы вводим в программу такую переменную, как **maxim**, предварительно присвоив ей маленькое значение (например, -1).

Таким образом, если значение ячейки **kol** превышает значение ячейки **maxim**, то находится самое длинное слово в строке при текущем проходе цикла. Следовательно, на выходе из цикла в ячейке **maxim** будет храниться количество символов в самом длинном слове строки.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
kol=0
maxim=-1
n=len(stroka)
for i in range(0, n):
    if stroka[i]!=" ":
        kol=kol+1
        if kol>maxim:
            maxim=kol
    else:
        kol=0
print("\n Самое длинное слово имеет ", maxim, "символов")
```

Результат работы программы представлен на рисунке 119.

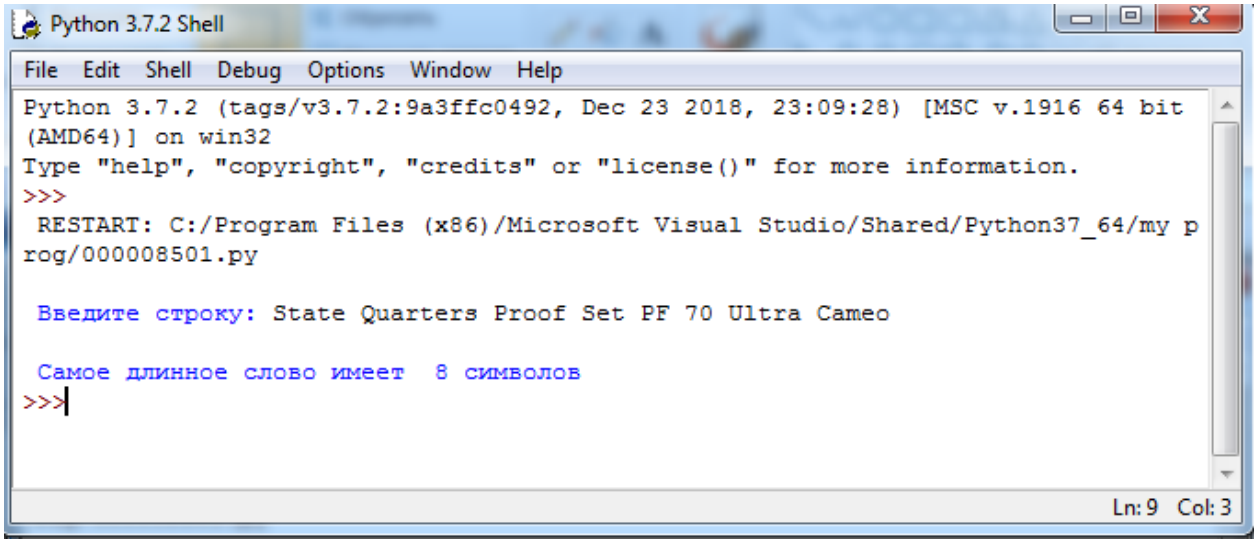

Рисунок 119 – Результат работы программы по нахождению длины самого большого слова в строке

Задача 8.5.2. Пользователь вводит строку **stroka**, содержащую несколько слов, разделенных запятой, но в некоторых местах пробел после запятой отсутствует. Результатом работы программы должна стать исправленная строка.

Решение. В переменной **k** хранится длина строки. В цикле с оператором **while** начинаем перебирать символы строки. В качестве условия поиска проверяем такую ситуацию, когда очередной символ строки равен символу «запятая», и при этом символ после запятой не равен пробелу. Как только условие становится истинным, вставляем пробел.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: ")
n=len(stroka)
probel=" "
i=1
stroka1=stroka[0]
while i<n:
    stroka1=stroka1+stroka[i]
    if (stroka[i]==","):
        if (stroka[i+1]!=" "):
            stroka1=stroka1+probel
    i=i+1
print("\n ", stroka1)
```

Результат работы программы представлен на рисунке 120.

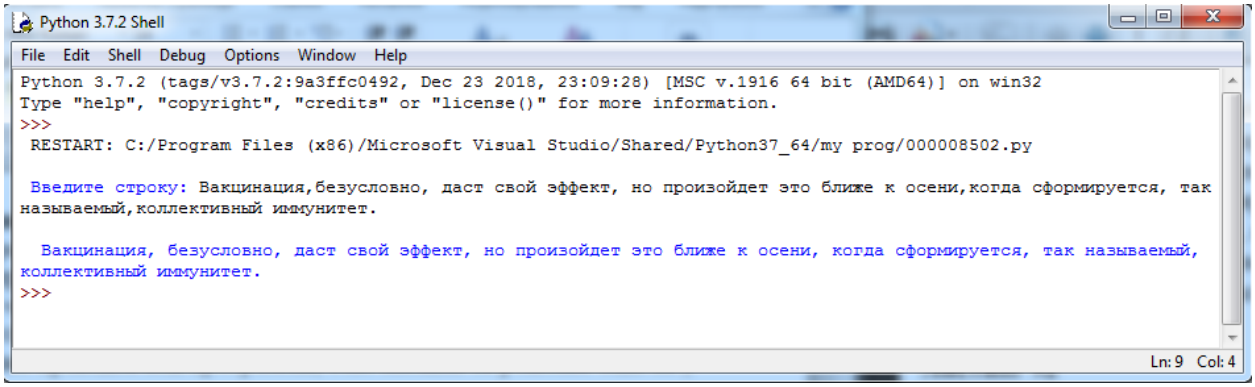

Рисунок 120 – Результат работы программы по исправлению строки

Задача 8.5.3. Пользователь вводит строку **stroka**, содержащую несколько слов и один из символов **simvol**, содержащихся в строке. Следует удвоить символ и вывести результирующую строку.

Решение. После ввода исходной строки осуществляем ее преобразование в список символов, используя функцию **list**. Оператором **zamena=simvol+simvol** удваиваем введенный символ **simvol**. В цикле с

оператором **for** выполним проверку равенства введенного символа символам исходной строки. Если условие будет истинным, такой символ найден, и можно его удвоить, выполнив оператор **spisok[i]=zamena**. Далее осуществляется преобразование списка в строку методом **join()** и вывод ее на экран.

Ниже приведен код программы, отвечающий за решение задачи.

```
stroka=input("\n Введите строку: \n")
spisok=list(stroka)
simvol=input("\n Введите символ, который необходимо удвоить: ")
zamena=simvol+simvol
n=len(spisok)
for i in range (0, n):
    if spisok[i]==simvol:
        spisok[i]=zamena
strokazam="".join(spisok)
print("\n Строка с удвоенным символом", simvol, " будет выглядить так: \n", strokazam)
```

Результат работы программы представлен на рисунке 121.

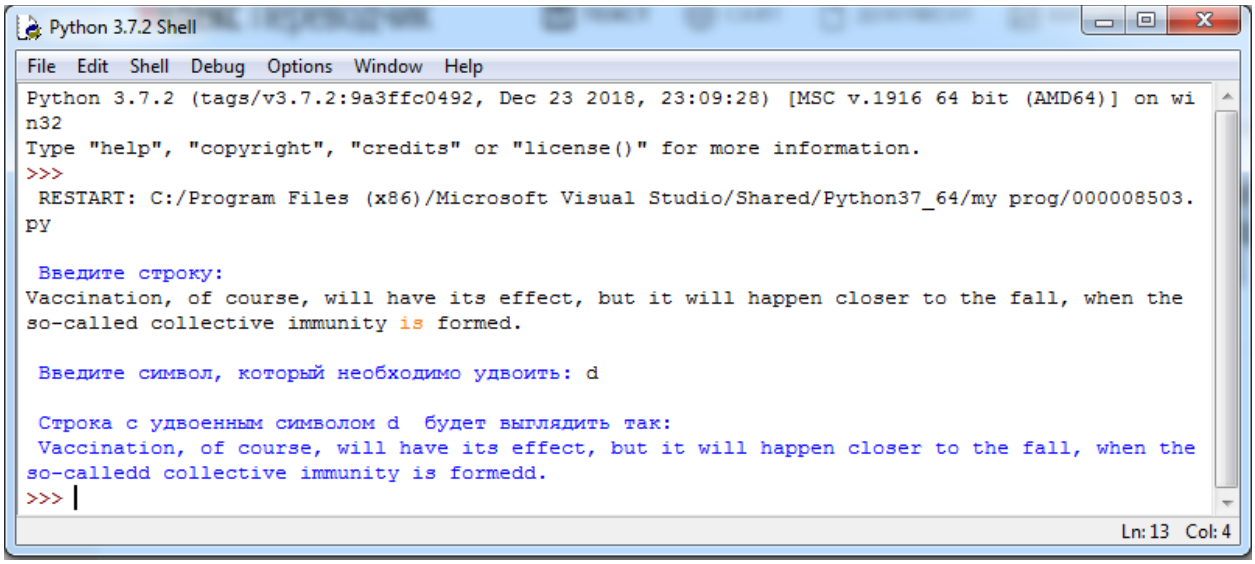

Рисунок 121 – Результат работы программы по удваиванию символов

Задача 8.5.4. Разработайте программу, проверяющую степень надежности пароля пользователя, при этом критерии сложности пароля следующие:

- ненадежный - длина менее 6 символов, либо длина от 6 до 8 символов, и пароль состоит только из цифр или букв в одном регистре;
- средний - длина от 6 до 8 символов, должен включать цифры и/или строчные и/или прописные символы, или иметь длину более 8 символов, но пароль при этом не содержит либо прописных букв, либо строчных, либо цифр;

- сложный - длина пароля свыше 8 символов, включает одновременно прописные буквы, строчные буквы и цифры.

Решение. В программе использованы различные методы работы со строками, такие как **isalpha()**, проверяющий, состоит ли строка из букв, **isdigit()**, проверяющий, состоит ли строка из цифр, **isupper()**, проверяющий, есть ли в строке символы в верхнем регистре, **islower()**, проверяющий, есть ли в строке символы в нижнем регистре.

Ниже приведен код программы, отвечающий за решение задачи.

```
print("\n Для создания надежного пароля используйте: \n 1. Заглавные буквы алфавита (от A до Z). \n 2. Строчные буквы алфавита (от a до z). \n 3. Цифры (от 0 до 9). \n 4. Длина пароля >8 символов")
parol=input("\n Введите пароль: ")
c="0123456789"
f1=False
f2=False
f3=False
for i in parol:
    if i.isupper():
        f1=True
    if i.islower():
        f2=True
    if i in c:
        f3=True
if len(parol)>=8:
    if f1==True:
        if f2==True:
            if f3==True:
                print("Ваш пароль ", parol, " очень надежный")
            else:
                print("Ваш пароль ", parol, " средней сложности, рекомендуем его сменить")
        else:
            print("Ваш пароль ", parol, " средней сложности, рекомендуем его сменить")
    else:
        print("Ваш пароль ", parol, " средней сложности, рекомендуем его сменить")
if 6<len(parol)<8:
    if parol.isalpha()==True:
        if (f1==False) or (f2==False):
            print("Ваш пароль ", parol, " ненадежен, срочно смените его")
    if parol.isdigit()==True:
        print("Ваш пароль ", parol, " ненадежен, срочно смените его")
```

```
        if f1==True:
            if f2==True:
                print("Ваш пароль ", parol, " средней сложности, рекомендуем
его сменить")
            else:
                if f3==True:
                    print("Ваш пароль ", parol, " средней сложности, рекомендуем
его сменить")
        elif f3==True:
            if f2==True:
                print("Ваш пароль ", parol, " средней сложности, рекомендуем
его сменить")
    if len(parol)<=6:
        print("Ваш пароль ", parol, " ненадежен, срочно смените его")
```

Результат работы программы представлен на рисунке 122.

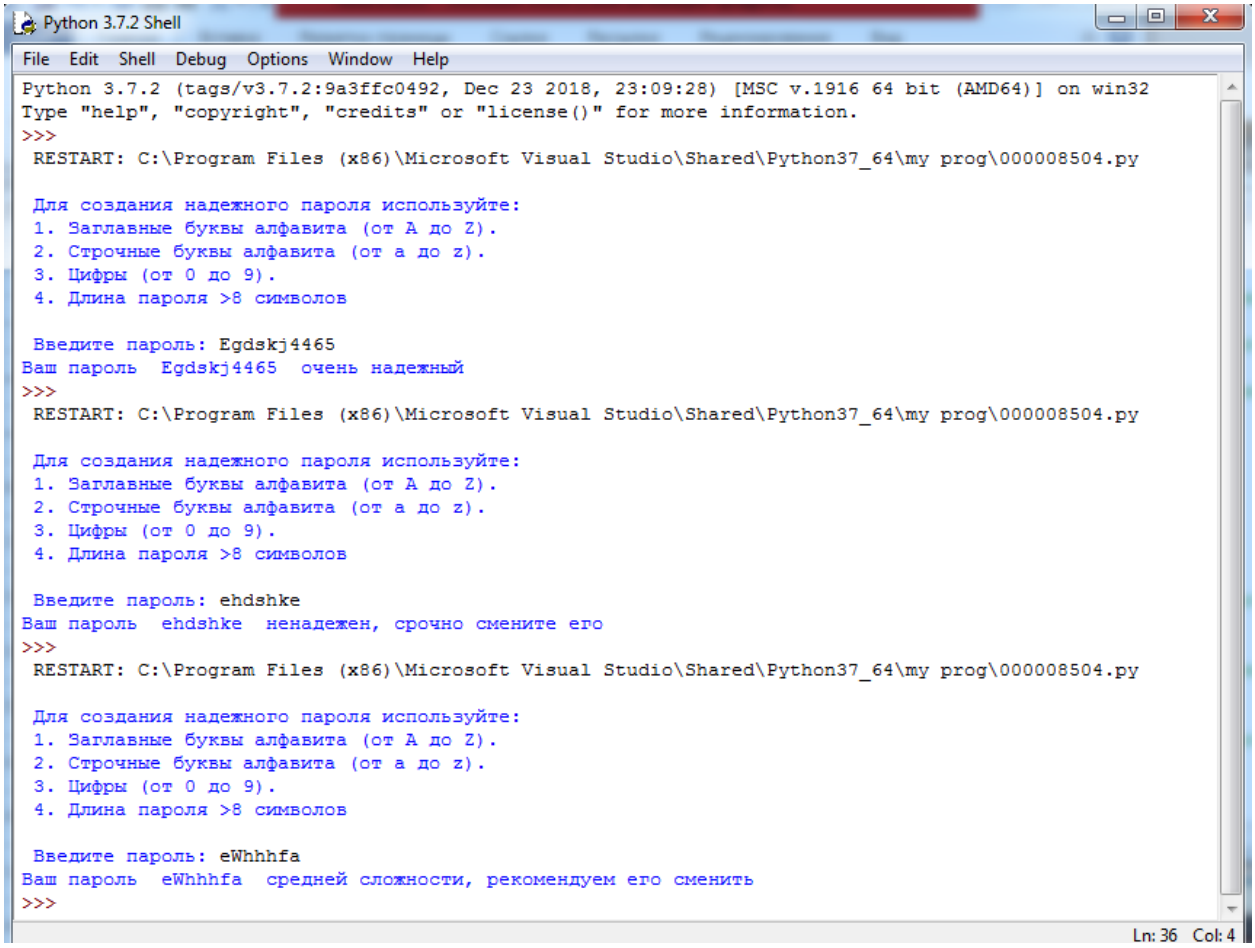

Рисунок 122 – Результат работы программы по проверки надежности пароля

Задача 8.5.5. Разработайте программу, которая подсчитывает число слов во введенной с клавиатуры строке. Примем, что словом считается любая последовательность символов, которая отделена от других пробелом.

Решение. Метод **capitalize()** переводит первый символ строки в верхний регистр, поэтому в результирующей строке произойдут изменения, если пользователь начнет набирать строку строчными буквами. С помощью метода **split(" ")**, который осуществляет разбиение строки по разделителю (в нашем случае - пробел), будут выявлены слова во введенной строке.

Ниже приведен код программы, отвечающий за решение задачи.

```
k=0
stroka=input("\n Введите предложение \n")
s=stroka.capitalize()
spisok=stroka.split(" ")
for i in range (len(spisok)):
    k=k+1
print("\n Предложение: ", s, "\n В этом предложении ", k, " слов(а)")
```

Результат работы программы представлен на рисунке 123.

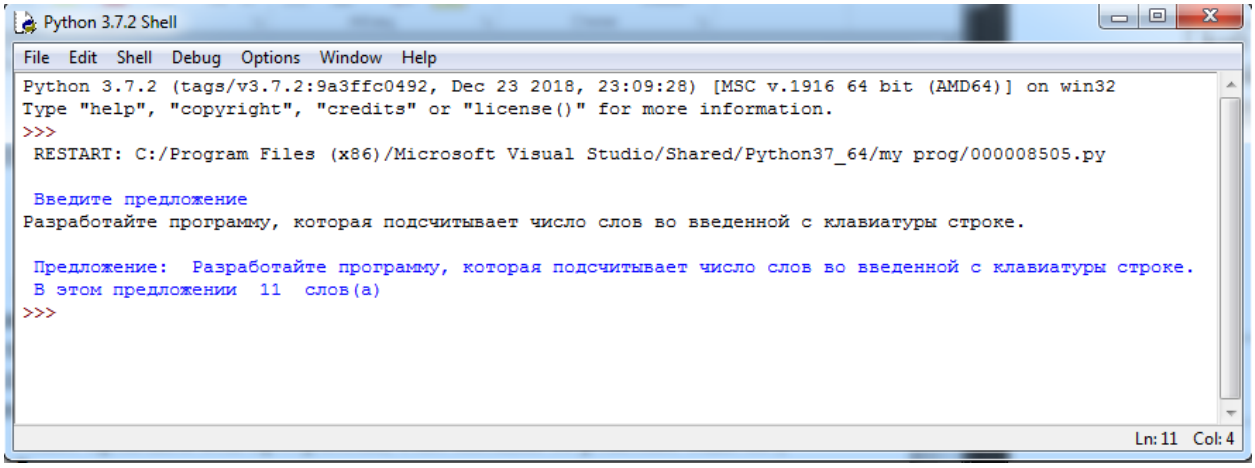

Рисунок 123 – Результат работы программы подсчета слов в тексте

Задача 8.5.6. Дана строка символов. Определите количество слов, являющихся записью натурального числа (состоят только из цифр).

Решение. Воспользуемся методом **isdigit()**, назначение которого состоит в анализе строки на предмет наличия в ней цифр. В цикле с оператором **for** будем проверять каждое слово строки и в случае истинности условия, увеличивать счетчик на единицу.

Ниже приведен код программы, отвечающий за решение задачи.

```
k=0
stroka=input("\n Введите строку: ")
spisok=stroka.split(" ")
n=len(spisok)
for i in range (n):
    if spisok[i].isdigit():
```

```
        k=k+1
stroka1="".join(stroka)
print("\n Количество слов, являющихся записью чисел ", k)
```

Результат работы программы представлен на рисунке 124.

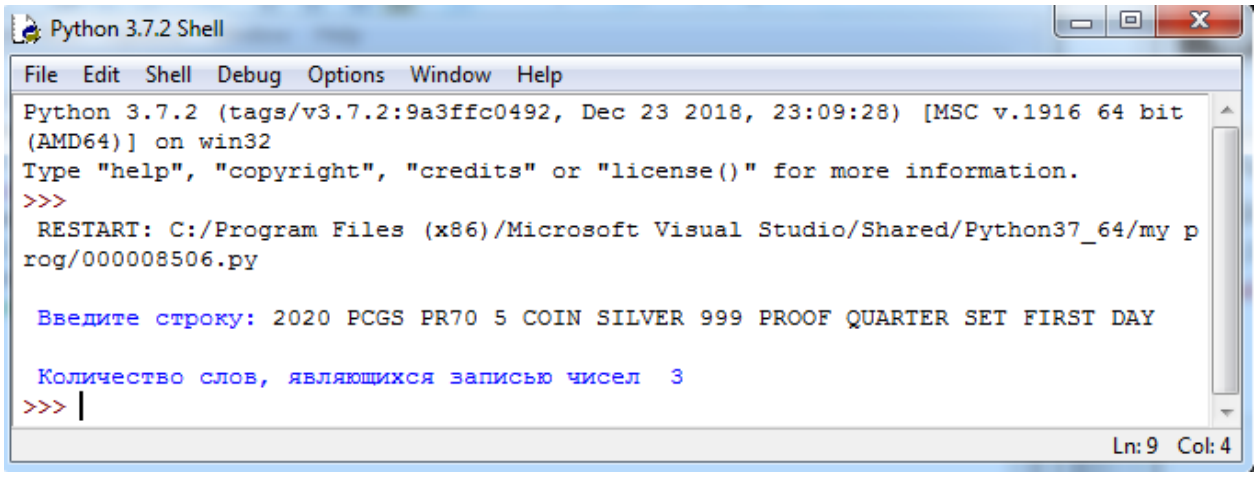

Рисунок 124 – Результат работы программы подсчета натуральных чисел в тексте

8.6 Контрольные вопросы

1. Как называется кодировка, поддерживающая кодирование буквенно-цифровых символов? Расскажите о ее структуре.
2. Перечислите основные функции для работы с символами. Приведите примеры.
3. Перечислите методы работы со строками, позволяющие преобразовывать символы строки к различным регистрам клавиатуры.
4. Какой метод позволяет разбить строку на подстроки? Напишите его синтаксис.
5. Какой метод отвечает за преобразование строки в список? Напишите его синтаксис.
6. Приведите примеры базовых алгоритмов строк.
7. Каким образом можно осуществить срез строки?

8.7 Задачи для самостоятельного решения

1. Разработайте программу, которая подсчитывает число слов во введенной с клавиатуры строке. Условимся считать словом любую последовательность символов, которая отделена от других пробелом.
2. Разработайте программу, которая проверяет, является ли введенная с клавиатуры последовательность символов целым числом, записанным в двоичной системе счисления.
3. Разработайте программу, которая вычисляет среднюю длину слов во введенной с клавиатуры строке.

4. Разработайте программу, которая зашифровывает введенный с клавиатуры текст. Процесс шифровки производится следующим образом: из десятичного кода каждого введенного с клавиатуры символа вычитается число десять. Получившаяся в результате вычитания величина интерпретируется как десятичный код некоторого другого символа, который и выводится на экран компьютера.

5. Дано слово. Определите, является ли оно палиндромом (словом, которое читается одинаково в обоих направлениях, например, «потоп»).

6. Дана строка символов. Определите самое длинное слово в строке и количество слов такой же длины.

7. Дана строка символов. Удалите из нее все пробелы.

8. Дана строка символов. Дано слово. Удалите из строки это слово.

9. Дана строка символов. Выделите подстроку между первой и второй точками.

10. Дана строка символов. Определите длину самого короткого и самого длинного слова.

11. Дана строка символов. Определите, сколько слов начинается и кончается одной и той же буквой.

12. Дана строка символов. Определите, сколько слов содержат хотя бы одну букву «е».

13. Дана строка символов. Определите, является ли она правильным скобочным выражением. Рассматривайте только круглые скобки.

14. Дана строка символов. Определите, сколько слов содержат ровно три буквы «е».

15. Строка содержит только цифры. Удалите все впереди стоящие нули.

16. Дана строка символов. Подсчитайте количество знаков препинания в строке.

17. Дана строка символов. Удалите из строки все запятые.

18. Дана строка символов. Приведено некоторое слово. Вставьте его после каждого пробела.

19. Дана строка символов. Найдите сумму чисел, встречающихся в строке.

20. Дана строка символов. Удалите из строки все числа.

21. Дана строка символов. Удалите из строки слово, имеющее наибольшую длину.

22. Дана строка символов. Вставьте в строку пробел после каждого знака препинания.

23. Дана строка из цифр и латинских букв. Определите, каких букв - гласных (A, E, I, O и т.д.) или согласных - больше в этой строке.

24. Из заданной строки удалите те ее части, которые заключены в кавычки (вместе с кавычками).

25.Задано слово. Замените в этом слове букву А на букву О. Если буквы А в этом слове нет, то выведите соответствующее сообщение.