Лабораторная работа №2. Списки и строки в Питоне

1. Цель и порядок работы

Цель работы – начальное изучение средств работы со списками и строками в языке Питон.

Порядок выполнения работы:

Самостоятельно проработать теоретические основы по языку программирования Питон, выполняя указанные в тексте примеры.

Выполнить индивидуальное задание.

Выполнение заданий можно осуществлять в среде интерпретатора, позволявшей пошаговое выполнение команд.

2. Краткая теория

2.1 Списки

Списки в Питоне являются одним из основных типов данных. Их используют для группирования вместе нескольких значений и обозначают, как список элементов, разделенных запятыми, заключенный в квадратные скобки.

2.1.1 Общее понятие о работе со списками

Создадим однородный список из целых чисел (тип integer):

```
>>> b = [100, 1, 3, 4, 1234]
>>>
```

Убедимся, что мы действительно его создали:

```
>>> b
[100, 1, 3, 4, 1234]
```

Аналогичным образом создадим однородный список из чисел с плавающей запятой (тип float):

```
>>> c = [1.0, 3.14, 2.71]
>>> c
[1.0, 3.1400000000000001, 2.71]
```

Далее создадим однородный список элементами которого будут являться строковые переменные

(тип string):

```
>>> rgb = ['red', 'green', 'blue']
>>> rgb
['red', 'green', 'blue']
```

Не обязательно, чтобы элементы списка были одного типа.

```
>>> a = ['cat', 'dog', 'fish', 1, 2, 33, 8.0]
>>> a
['cat', 'dog', 'fish', 1, 2, 33, 8.0]
```

2.1.2 Нумерация элементов списка. Срезы. Многомерные списки.

Нумерация элементов

Все элементы списка пронумерованы. В общепринятой терминологии номер элемента списка называется индексом. Условимся называть начало списка (самый левый элемент) "головой", а конец (самый правый элемент) - "хвостом".

Нумерация элементов списка начинается с нуля. Например, для списка a из предыдущего раздела:

```
>>> a
['cat', 'dog', 'fish', 1, 2, 33, 8.0]
>>> a[0]
'cat'
>>> a[2]
'fish'
>>> a[3]
1
```

Номер элемента списка может быть и отрицательным. В этом случае - элемент "минус один" - это хвостовой, последний элемент списка, "минус два" - это предпоследний, "второй с хвоста" элемент:

```
>>> a
['cat', 'dog', 'fish', 1, 2, 33, 8.0]
>>> a[-1]
8.0
>>> a[-2]
33
```

К каждому элементу списка можно обратится по его индексу:

```
>>> b
[100, 1, 3, 4, 1234]
```

```
>>> b[0] = 1
>>> b
[1, 1, 3, 4, 1234]
>>> b[3] = b[4]
>>> b
[1, 1, 3, 1234, 1234]
```

Срезы.

Список может быть разделен на несколько частей. Это делается с помощью так называемых "срезов". Для того, чтобы осуществить срез следует указать название списка и в квадратных скобках после него указать начальный и конечный элемент среза:

```
>>> a
['cat', 'dog', 'fish', 1, 2, 33, 8.0]
>>> a[1:4]
['dog', 'fish', 1]
```

При срезе списка удобно мысленно нумеровать не элементы, а промежутки (интервалы) между ними. Это очень удобно для указания произвольных срезов. Перед нулевым (по индексу) элементом списка промежуток имеет номер 0, после него - 1 и так далее. Отрицательные значения отсчитывают промежутки с конца строки.

```
>>> a
['cat', 'dog', 'fish', 1, 2, 33, 8.0]
>>> a[2:3] ['fish']
>>> a[2:2]
[]
>>> a[2:5] ['fish', 1, 2]
```

Обратите внимание на то, что для a[2:2] был получен пустой список (это логично - если срез списка начинается и заканчивается в одном и том же интервале - это пустой список).

Если до или после двоеточия не указывается индекс, интерпретатор Питона истолковывает это как указание "до конца списка". Пример:

```
>>> a[:5]
['cat', 'dog', 'fish', 1, 2]
>>> a[3:]
[1, 2, 33, 8.0]
```

Вложенные (многомерные) списки

Список допускает, что его элементом может быть другой (вложенный) список:

```
>>> m = [1, 2, ['a', 'b', 'c'], 3, 8, 14]
>>> m[0]
1
>>> m[1]
2
>>> m[2]
['a', 'b', 'c']
>>> m[3]
3
```

Обратите внимание, что m[2] - это список ['a', 'b', 'c'] целиком.

Как обратится к одному из элементов этого вложенного списка? Нужно указать еще один, дополнительный индекс:

```
>>> m
[1, 2, ['a', 'b', 'c'], 3, 8, 14]
>>> m[2]
['a', 'b', 'c']
>>> m[2][0]
'a'
>>> m[2][1]
'b'
>>> m[2][:2] ['a', 'b']
```

2.1.3 Действия со списками

Списки допускают два типа действий - конкатенацию и размножение (мультипликацию). Конкатенация обозначает склейку списков и осуществляется следующим образом:

```
>>> s1 = [1, 2, 3, 4]
>>> s2 = [10, 20, 30, 40]
>>> s3 = ['a', 'b', 'c']
>>> sk = s1 + s2 + s3 + [2, 2]
>>> sk
[1, 2, 3, 4, 10, 20, 30, 40, 'a', 'b', 'c', 2, 2]
```

Размножение списков осуществляется с помощью знака умножения:

```
>>> s1
[1, 2, 3, 4]
>>> sm = s1 * 4
>>> sm
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

2.1.4 Методы работы со списками

Длина списка

Функция len(список) - определяет длину списка, указанного в качестве аргумента:

```
>>> sp = [1, 3, 3, 4, 5, 67, 7, 1, 1]
>>> sp
[1, 3, 3, 4, 5, 67, 7, 1, 1]
>>> len(sp)
9
>>> sp[1:]
[3, 3, 4, 5, 67, 7, 1, 1]
>>> len(sp[1:])
8
>>> sp[:4]

[1, 3, 3, 4]
>>> len(sp[:4])
4
>>> sp[1:4] [3, 3, 4]
>>> len(sp[1:4])
3
```

Генерация списка с последовательными номерами

Функция range(N) отвечает за генерацию списка из N элементов арифметической прогрессии. Напоминаем: счет элементов начинается с нуля:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(20)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> range(40)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39]
```

Также можно указать начальный элемент прогрессии:

```
>>> range(5,10) [5, 6, 7, 8, 9]
>>> range(5,20)
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> range(15,40)
[15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
```

Также можно указывать начальный элемент и шаг прогрессии:

```
>>> range(5,10,2) [5, 7, 9]
>>> range(15,40,5)
[15, 20, 25, 30, 35]
```

Обратите внимание: часто функция range() используется в связке с функцией len() - чтобы создать нумерованную последовательность списка. Это может значительно упрощать работу с обработкой списков:

```
>>> sk
[1, 2, 3, 4, 10, 20, 30, 40, 'a', 'b', 'c', 2, 2]
>>> range(len(sk))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

В дальнейшем будут приведены примеры использования такой связки в циклах обработки списков.

Добавление элемента к списку

Присоединение элемента к списку осуществляется с помощью функции append(). Обратите внимание на то, как эта функция используется в приведенных ниже примерах:

```
>>> k = [1, 2]
>>> k.append(3)
>>> k
[1, 2, 3]
>>> k.append(3)
>>> k
[1, 2, 3, 3]
>>> k.append(9)
>>> k

[1, 2, 3, 3, 9]
```

Сортировка списка

Сортировка списка осуществляется с помощью функции sort() и reverse(), которые отсортировывают список по порядку:

```
>>> sk
[1, 2, 3, 4, 10, 20, 30, 40, 'a', 'b', 'c', 2, 2]
>>> sk.sort()
>>> sk
[1, 2, 2, 2, 3, 4, 10, 20, 30, 40, 'a', 'b', 'c']
>>> sk.reverse()
>>> sk
['c', 'b', 'a', 40, 30, 20, 10, 4, 3, 2, 2, 2, 1]
```

Добавление элементов в список

Существует специальная функция insert(i,n), которая добавляет элемент n в интервал i (если вы забыли что такое интервал - перечитайте раздел "Срезы"):

```
>>> a = [1, 1, 2, 2, 6, 13, 1, 23, 2, 7]
>>> a.insert(1,7)
>>> a
[1, 7, 1, 2, 2, 6, 13, 1, 23, 2, 7]
>>> a.insert(3,7)
>>> a
[1, 7, 1, 7, 2, 2, 6, 13, 1, 23, 2, 7]
>>> a.insert(3,'a')
>>> a
[1, 7, 1, 'a', 7, 2, 2, 6, 13, 1, 23, 2, 7]
```

Также можно обойтись и без этой функции - используя срезы. Вставим во второй интервал списка a строковую переменную 'b':

```
>>> a
[1, 7, 1, 'a', 7, 2, 2, 6, 13, 1, 23, 2, 7]
>>> a = a[:2] + ['b'] + a[2:]
>>> a
[1, 7, 'b', 1, 'a', 7, 2, 2, 6, 13, 1, 23, 2, 7]
```

Обратите внимание: 'b' вставляется в список не само по себе, а виде отдельного элемента- списка ['b']. В противном случае можно получить ошибку:

```
>>> a = a[:2] + 'b' + a[2:]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "str") to list
```

Удаление элементов из списка

За удаление элементов из списка отвечает команда del. Она может использоваться как для удаления одного из элементов, так и для удаления целых фрагментов списка, которые задаются с помощью срезов:

```
>>> a = [7, 'b', 1, 'a', 7, 2, 2, 6, 13, 1, 23, 2, 7]
>>> a
[7, 'b', 1, 'a', 7, 2, 2, 6, 13, 1, 23, 2, 7]
>>> del a[0]
>>> a
['b', 1, 'a', 7, 2, 2, 6, 13, 1, 23, 2, 7]

>>> len(a)
12
>>> del a[10:]
>>> a
['b', 1, 'a', 7, 2, 2, 6, 13, 1, 23]
>>> del a[4:6]
>>> a
['b', 1, 'a', 7, 6, 13, 1, 23]
```

Подсчет элементов в списке

Функция count() производит подсчет элементов в списке:

```
b = [1, 6, 3, 3, 4, 9, 2, 3, 4]
b.count(3) 3
b.count(1) 1
b.count(4) 2
b.count(-1) 0
```

В последнем случае был задан элемент, который отсутствует в списке - поэтому в качестве результата был получен ноль.

Поиск вхождения в список первого элемента

За поиск вхождения в список первого элемента отвечает функция index():

```
>>> b = [1, 6, 3, 3, 4, 9, 2, 3, 4]
```

```
>>> b.index(3)
2
>>> b.index(9)
5
```

Если элемент отсутствует - будет выдана ошибка:

```
>>> b.index(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.index(x): x not in list
```

Удаление из списка элементов с заданным значением

Функция remove() удаляет из списка первое вхождение элемента с заданным значением:

```
>>> b = [1, 6, 3, 3, 4, 9, 2, 3, 4]
>>> b
[1, 6, 3, 3, 4, 9, 2, 3, 4]
>>> b.remove(3)
>>> b
[1, 6, 3, 4, 9, 2, 3, 4]
>>> b.remove(3)
>>> b
[1, 6, 4, 9, 2, 3, 4]
>>> b.remove(9)
>>> b
[1, 6, 4, 2, 3, 4]
```

2.2 Строки

2.2.1 Общее понятие о работе со строками

Строковые переменные в языке Питон обозначаются одинарной либо двойной кавычками. Можно использовать любое из этих обозначений:

```
>>> s = 'string 1'
>>> s
'string 1'
>>> s = "string 1"
>>> s
'string 1'
>>> s = "string 2"
>>> s
```

'string 2'

Если переменная занимает больше одной строки - можно использовать для ее обозначения тройные кавычки. В приведенном ниже примере троеточия не вводятся:

```
>>>  s =  ' ' '
... first  line
... second line
... third  line
... ' ' '
>>>  s
'\nfirst  line\nsecond  line\nthird  line\n'
```

Обратите внимание, что введенная переменная вытянута в одну линию, разбитую на абзацы с помощью спецсимвола \n, обозначающего конец строки. Чтобы посмотреть переменную "с абзацами" следует использовать команду print:

```
>>>  print  s

first  line
second line
third  line
```

Когда нужно набрать очень длинную одиночную строку - можно использовать символ "бэкслэш" -
\ - "обратный слэш":

```
>>>  x  =  'this  is \
... multilong\
... string  \
... but  it \
... may  view  \
... as  one line'
>>>  x
'this  is multilongstring  but  it may  view  as  one line'
```

Обратите внимание на местонахождение пробела перед обратным слэшем.

Спецсимволы

С одним из спецсимволов мы уже сталкивались в предыдущем разделе - это \n - символ перевода строки. Так же очень распространенными символами

являются \t - символ табуляции, \r - символ возврата каретки, \a - bell (звонок). Обратите внимание на то, что чтобы увидеть символы "в действии", нужно распечатать переменную с помощью команды print, а в обычном режиме обращения спецсимволы показываются как они есть - чтобы ими было удобно манипулировать:

```
>>> z = 'new line\ndemo'
>>> z
'new line\ndemo'
>>> print z
new line
demo
>>> z = 'tabulation\tdemo'
>>> z
'tabulation\tdemo'
>>> print z

tabulation    demo
```

Экранирование спецсимволов

Если необходимо оперировать со строками в которых есть символ "бэкслэш" - это может вызвать проблемы. Например, такая ситуация может возникать, при указании путей в файловых системах fat32 или ntfs (используемых в семействе ОС MS Windows).

```
>>> path = "D:\temp\new_file.txt"
>>> path
'D:\temp\new_file.txt'
>>> print path
D:      emp
ew_file.txt
```

Здесь интерпретатор Питона посчитал кусочки строк \t и \n - символами табуляции и новой строки соответственно. Что привело к ошибкам.

Чтобы такого не происходило - бэкслэш необходимо экранировать. Это делается с помощью второго бэкслэша, который экранирует использование первого. Таким образом, если для питона \n - это символ new line ("новая строка"), то \\n - это просто символ бэкслэша, за которым следует символ n:

```
>>> back = 'new line\ndemo'
>>> back
'new line\ndemo'
>>> print back
new line
```

```
demo
>>> back = 'new line\\ndemo'
>>> back
'new line\\ndemo'
>>> print back
new line\ndemo
```

Аналогично это действует для других спецсимволов:

```
>>> back = 'tabulation\tdemo'
>>> back
'tabulation\tdemo'
>>> print back
tabulation    demo
>>> back = 'tabulation\\tdemo'
>>> back
'tabulation\\tdemo'
>>> print back
tabulation\tdemo
```

Поэтому при указании путей в ОС MS Windows следует использовать двойной слэш:

```
>>> path = "D:\\temp\\new_file.txt"
>>> path
'D:\\temp\\new_file.txt'
>>> print path
D:\temp\new_file.txt
```

Здесь все в порядке.

Есть еще один способ. Можно поставить перед кавычками строки строчную букву r - например так: r"D:\temp\new_file.txt". Это указание для Питона воспринимать содержимое кавычек в формате raw - как есть, без спецсимоволов:

```
>>> path = r"D:\temp\new_file.txt"

>>> path
'D:\\temp\\new_file.txt'
>>> print path
D:\temp\new_file.txt
```

Эти нюансы работы со строками будут важны в дальнейшем, когда мы перейдем к операциям с файлами и работой в файловой системе.

Юникод

Строчная u перед кавычками, обозначающими строку, указывает на то, что эта строка задана в
Юникоде (подробнее этот вопрос будет рассмотрен в дальнейшем).

2.2.2 Нумерация симоволов строки. Срезы

Нумерация строк в целом аналогична нумерации элементов списка. Только вместо элементов списка в строке нумеруются символы.

```
>>> br = 'lazy brown fox'
>>> br[0]
'l'
>>> br[2]
'z'
>>> br[5]
'b'
>>> br[4]
' '
>>> br[-1]
'x'
>>> br[-2]
'o'
>>> br[-3]
'f'
```

Обратите внимание на то, что спецсимволы \n, \t и так далее - с точки зрения интерпретатора являются одиночными символами:

```
>>> br = 'lazy\nbrown\tfox'
>>> br
'lazy\nbrown\tfox'
>>> print br
lazy
brown	fox
>>> br[3]
'y'
>>> br[4]
'\n'
>>> br[5]
'b'
>>> br[9]
'n'
```

```
>>> br[10]
'\t'
>>> br[11]
'f'
```

Работа со срезами, так же аналогична тому как это организовано у списков, за тем исключением, что нумеруются не интервалы между списками, а интервалы между символами строки:

```
>>> br = 'lazy brown fox'
>>> br[5:]
'brown fox'
>>> br[:4]

'lazy'
>>> br[5:10]
'brown'
>>> br[-3:]
'fox'
```

2.2.3 Работа со строками. Склейка и размножение

Работа со строками в целом аналогична работет со списками, за одним исключением - строковые переменные нельзя менять посимвольно. Вместо этого можно использовать срезы и операции склейки.

Строки допускают те же операции, что и списки. С ними можно производить операции склейки:

```
>>> z = "help"
>>> y = "me"
>>> x = z + y
>>> x
'helpme'
>>> x = z + " " + y
>>> x
'help me'
```

Аналогично происходить операция размножения:

```
>>> z * 3
'helphelphelp'
>>> y * 10
'memememememememememe'
>>> x = z + " "*7 + y
```

```
>>> x
'help  me'
>>> x = (z + " ") * 7 + y
>>> x
'help help help help help help help me'
```

Хотя мы не можем изменить отдельный элемент строки:

```
>>> br = 'lazy brown fox'
>>> br[1] = 'z'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Мы можем работать со склейкой и срезами так, чтобы добиться нужного эффекта:

```
>>> br
'lazy brown fox'
>>> br = br[:1]+'z'+br[2:]
>>> br
'lzzy brown fox'
```

2.2.4 Функции работы со строками

Длина строки

Функция len() работает аналогично функции len() для списков:

```
>>> br = 'lazy brown fox'
>>> len(br)
14
```

Переход от числовой переменной к строке и обратно

Функции str() (переход к символьному виду), float() (переход к числу с плавающей запятой)
и int() (переход к целому числу) обеспечивают операции переходов между типами:

```
>>> a = '2010'
>>> a
'2010'
>>> int(a)
2010
```

```
>>> float(a)
2010.0
>>> pi = 3.1415926
>>> pi
3.1415926000000001
>>> str(pi)
'3.1415926'
>>> int(pi)
3
>>> year = 2010
>>> str(year)
'2010'
>>> float(year)
2010.0
```

Получение ascii-кодов символов

Функция ord() обеспечивает получение ascii-кода символа, chr() - получение символа по его ascii-коду:

```
>>> print chr(34)
"
>>> print chr(36)
$
>>> print chr(37)
%
>>> ord('$')
36
>>> ord('@')
64
>>> ord('f')
102
>>> ord('z')
122
```

Ввод с клавиатуры

Функция raw_input('приветствие') останавливает выполнение программы, печатает приветствие и ждет ввода с клавиатуры, завершающегося нажатием <Enter>. При этом функция возвращает набранное пользователем в виде строки:

```
>>> a = raw_input('Input please: ')
Input please: Please
>>> a
```

'Please'

Как правило используется в сочетании с функцией int() или float(), чтобы сразу преобразовать введенное число в нужный тип:

```
>>> a = int(raw_input('Input integer number: '))
Input integer number: 888
>>> a

888
>>> f = float(raw_input('Input real number: '))
Input real number: 3.14
>>> f
3.1400000000000001
```

3. Варианты заданий

3.1 Списки

Вариант студента определяется по последней цифре в списке студентов в группе.

Задайте переменную spisok, в которую занесите любые семь элементов из следующего набора данных (выбирается согласно варианту):

1. Список дней недели.
2. Название любых семи фильмов.
3. Список из семи любых названий месяцев.
4. Название любых семи стран.
5. Список из семи любых названий предметов в расписании.
6. Список из семи любых предметов в вашем рюкзаке.
7. Название любых семи улиц города.
8. Название любых семи городов.
9. Название любых семи марок автомобилей.

0.Название любых семи деталей компьютера.

Общее задание для всех вариантов:

• Распечатайте первый, третий, четвертый, последний и предпоследний элементы списка.

• Разбейте с помощью срезов список на три части - в первой части три элемента, во второй - один, в третьей - тоже три.

• Создайте новый список mnogo_spisok, в который бы в виде элементов входили три части из предыдущего задания (то есть первый элемент mnogo_spisok - это под-список из первых трех элементов списка spisok, второй

элемент - это один, средний элемент списка spisok и так далее)

• Поменяйте местами первый и последний элементы mnogo_spisok

• Добавьте к mnogo_spisok еще один, любой элемент

• Добавьте еще один, любой элемент к любому из подсписков mnogo_spisok

• Удалите из списка mnogo_spisok все элементы кроме первого

3.2 Строки

Варианты заданий определяться по номеру в списке студентов:
1 вариант 1,21 задание.
2 вариант 2,22 задание.
3 вариант 3,23 задание.
4 вариант 4,24 задание.
5 вариант 5,25 задание.
6 вариант 6,26 задание.
7 вариант 7,27 задание.
8 вариант 8,28 задание.
9 вариант 9,29 задание.
10 вариант 10,30 задание.
11 вариант 11,31 задание.
12 вариант 12,32 задание.
13 вариант 13,33 задание.
14 вариант 14,34 задание.
15 вариант 15,35 задание.
16 вариант 16,36 задание.
17 вариант 17,37 задание.
18 вариант 18,38 задание.
19 вариант 19,39 задание.
20 вариант 20,40 задание.

1) Написать программу, которая преобразует строку таким образом, что цифры, которые находятся в слове, переносятся в конец строки без изменения порядка следования остальных символов.
2) Написать программу, которая, если строка начинается и оканчивается одним и тем же знаком, во всей строке заменяет этот знак четвертым символом строки.
3) Дана строка состоящая из слов, разделенных пробелами (одним ил несколькими). Вывести строку содержащую эти же слова (разделенные одним пробелом), но расположенные в обратном порядке.
4) Дана строка символов. Для сохранения ее в сжатом виде найти максимальную последовательность символов произвольной длины, которая повторяется, и заменить ее своим кодом.

5) В строку S добавить необходимое количество пробелов так, чтобы ее длина стала равна n. Причем: перед первым словом пробелы не добавлять, после последнего слова все пробелы удалить, добавленные пробелы равномерно распределить между словами. Если длина S превосходит n, удалить S из все слова, которые не укладываются в первые n символов, а оставшуюся часть преобразовать по вышеуказанным правилам.
6) Палиндромом называют последовательность символов, которая читается как слева направо, так и справа налево. Найти во введённой строке подстроку-палиндром максимальной длины.
7) Вывести сообщение "МОЖНО", если из букв введённой строки X можно составить введенную строку Y, при условии, что каждую букву строки X можно использовать один раз; и сообщение "НЕЛЬЗЯ" в противном случае.
8) По введённому числовому значению N (0 < N < 4000) вывести его запись в римской системе счисления. Римская система счисления использует 7 цифр (I=1 V=5 X=10 L=50 C=100 D=500 M=1000).
9) Дана строка символов. Найти слово, в котором число различных символов минимально. Если таких слов несколько, найти первое из них.
10) Дана строка символов. Найти количество слов, содержащих только символы латинского алфавита, а среди них – количество слов с равным числом гласных и согласных букв.
11) Дана строка символов. Найти слово, символы в котором идут в строгом порядке возрастания их кодов. Если таких слов несколько, найти первое из них.
12) Дана строка символов. Найти слово, состоящее только из различных символов. Если таких слов несколько, найти первое из них.
13) Заменить в данной строке все вхождения подстроки s на порядковый номер вхождения. Подстрока s вводится с клавиатуры.
14) Ьтасипан уммаргорп адовереп йоннадаз икортс оп умещюуделс упицнирп.
15) В строке переставить местами рядом стоящие слова.
16) Из строки вырезать слова, стоящие на четном месте.
17) В строке удалить все пробелы, а затем после каждой пятой буквы вставить знак вопроса.
18) Переставить местами слова в строке (первая с последней и т.д.).
19) Из строки удалить все встречающиеся символы.
20) Все слова в строке расположить в алфавитном порядке.
21) Подсчитать сколько раз в данной строке встречается некоторая последовательность букв, введенная с клавиатуры.
22) В строке после каждого слова вставить запятую.
23) Каждое слово в строке распечатать с новой строчки экрана.
24) В строке удалить последнюю букву у слов.
25) В строке все запятые заменить точкой, и перед первым словом вставить слово STRING.

26) Подсчитать количество слов и букв в этих словах в строке.
27) В строке удалить все знаки препинания.
28) С клавиатуры считывается строка состоящая из цифр от 0 до 9. Разбить ее на две части, полученные строки преобразовать к целочисленному типу.
29) В строке каждый символ заменить на соответствующий ему код.
30) В строке удалить каждый заданный символ, а остальные продублировать.
31) В строке посчитать наибольшее количество идущих подряд пробелов.
32) В строке удалить лишние пробелы.
33) В строке подсчитать количество слов начинающихся с заданной буквы.
34) В строке удалить все символы, не являющиеся буквами или цифрами.
35) В строке найти самое длинное симметричное слово.
36) В строке оставить только те символы, которые встречаются один раз.
37) В строке указать слово, в котором количество гласных букв минимально.
38) В строке указать слово, в котором количество согласных букв максимально.
39) В строке удалить все заданные группы букв.
40) В строке подсчитать количество слов

4. Рекомендуемая литература

1. Васильев А. Н. Python на примерах. Практический курс по программированию. - СПб.: Наука и Техника, 2016. - 432 с.: ил.
2. http://python.org Официальный сайт языка Питон.
3. http://www.python.ru Русскоязычная документация языка Питон.
4. http://ru.wikipedia.org/wiki/Python Статья, посвященная Питону в Википедии.