```
break                # 5
```

В первой строке программы в переменную s считывается строка с клавиатуры. Затем организовывается цикл, в котором переменная s меняется от 0 до len(s)-1, то есть переменная i принимает подряд все номера символов в строке. В третьей строке программы проверяется, является ли i-й символ строки пробелом (то есть совпадает ли он со строкой из одного пробела). Если проверяемое условие истинно, то был найден первый слева строки пробел, на экран печатается первые i символов строки, то есть все символы с начала строки до найденного пробела, после чего выполнение цикла завершается инструкцией break.

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

Благодаря поддержке стандарта Unicode Python 3 может содержать символы любого языка мира, а также многие другие символы. Необходимость работы с этим стандартом была одной из причин изменения Python 2.

Строки являются первым примером последовательностей в Python. В частности, они представляют собой последовательности символов.В отличие от других языков, в Python строки являются неизменяемыми. Вы не можете изменить саму строку, но можете скопировать части строк в другую строку, чтобы получить тот же эффект.

Создаем строки с помощью кавычек

Строка в Python создается заключением символов в одинарные или двойные кавычки, как показано в следующем примере:

```
>>> 'Печенюшки'
'Печенюшки'
>>> "Семки"
'Семки'
```

Интерактивный интерпретатор выводит на экран строки в одинарных кавычках, но все они обрабатываются одинаково.Зачем иметь два вида кавычек? Основная идея заключается в том, что вы можете создавать строки, содержащие кавычки. Внутри одинарных кавычек можно расположить двойные и наоборот.

Для задания строк можно использовать тройные кавычки, это удобно для создания многострочного блока текста:

```
>>> poem = '''Товарищ, верь, пройдёт она,
и демократия, и гласность.
И вот тогда госбезопасность
Припомнит наши имена!'''
```

(Это стихотворение было введено в интерактивный интерпретатор, который поприветствовал нас символами >>> в первой строке и выводил символы … до тех пор, пока мы не ввели последние тройные кавычки и не перешли к следующей строке.)

Если бы вы попробовали создать стихотворение с помощью одинарных кавычек, Python выдал бы ошибку, когда бы вы перешли к следующей строке.

Если внутри тройных кавычек располагается несколько строк, символы конца строки будут сохранены внутри нее. Если перед строкой или после нее находятся пробелы, они также будут сохранены.

Вам может понадобиться работать с пустой строкой. В ней нет символов, но она совершенно корректна. Вы можете создать пустую строку с помощью любых упомянутых ранее кавычек:

```
>>> ''
''
>>> ""
''
>>> ''''''
''
>>> """"""
''
```

Зачем может понадобиться пустая строка? Иногда приходится компоновать строку из других строк и для этого нужно начать с чистого листа, то есть с пустой строки.

```
>>> men = 15
>>> base = ''
>>> base+=str(men)
>>> base+=' человек на сундук мертвеца'
>>> base
'15 человек на сундук мертвеца'
```

Преобразование типов данных с помощью функции str()

Вы можете преобразовывать другие типы данных Python в строки с помощью функции str():

```
>>> str(98.6)
'98.6'
>>> str(1.0e4)
'10000.0'
>>> str(True)
'True'
```

***Создаем управляющие символы с помощью символа ***

Python позволяет вам создавать управляющие последовательности внутри строк, чтобы добиться эффекта, который по-другому было бы трудно выразить. Размещая перед символом обратный слеш (\), вы наделяете этот символ особым значением. Наиболее распространена последовательность \n, которая означает переход на новую строку. С ее помощью вы можете создать многострочные строки из однострочных. Наберите в командной строке EDLE:

```
>>> palindrome = 'Аргентина \nманит \nнегра'
>>> print(palindrome)
Аргентина
манит
негра
```

Вы также увидите последовательность \t (табуляция), которая используется для выравнивания текста:

```
>>> print('\tКрасная шапочка')
        Красная шапочка
>>> print('Красная \tшапочка')
Красная         шапочка
>>> print('Красная шапочка\t')
Красная шапочка
```

В последней строке табуляция стоит в конце, ее вы, конечно, увидеть не можете.

Если вам нужен обратный слеш, просто напечатайте два:

```
>>> speech = 'Сегодня нам понадобился обратный слеш: \\'
>>> print(speech)
Сегодня нам понадобился обратный слеш: \
```

Объединяем строки с помощью символа +

Вы можете объединить строки или строковые переменные в Python с помощью оператора +, как показано далее:

```
>>> 'Я обернулся посмотреть '+'не обернулась ли она'
'Я обернулся посмотреть не обернулась ли она'
```

Можно также объединять строки (не переменные), просто расположив одну перед другой:

```
>>> "чтоб посмотреть, " "не обернулся ли я"
'чтоб посмотреть, не обернулся ли я'
```

Не забывайте добавлять пробелы при объединении строк.

***Размножаем строки с помощью символа **

Оператор * можно использовать для того, чтобы размножить строку. Попробуйте ввести в интерактивный интерпретатор следующие строки и посмотреть, что получится:

```
>>> start = 'Раз-два '*4 + '\n'
>>> end = 'Проверка связи.'
>>> print(start + end)
Раз-два Раз-два Раз-два Раз-два
Проверка связи.
```

Извлекаем символ с помощью символов [ ]

Для того чтобы получить один символ строки, задайте смещение внутри квадратных скобок после имени строки. Смещение первого (крайнего слева) символа равно 0, следующего — 1 и т. д. Смещение последнего (крайнего справа) символа может быть выражено как –1, поэтому вам не придется считать, в таком случае смещение последующих символов будет равно –2, –3 и т. д.:

```
>>> letters = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
>>> letters[0]
'а'
>>> letters[1]
'б'
>>> letters[-1]
'я'
>>> letters[-2]
'ю'
```

Если вы укажете смещение, равное длине строки или больше (помните, смещения лежат в диапазоне от 0 до длины строки –1), сгенерируется исключение:

```
>>> letters[33]
Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    letters[33]
IndexError: string index out of range
```

Поскольку строки неизменяемы, вы не можете вставить символ непосредственно в строку или изменить символ по заданному индексу. Попробуем изменить слово Маша на слово Даша и посмотрим, что произойдет:

```
>>> name = 'Маша'
>>> name[0] = 'Д'
Traceback (most recent call last):
  File "<pyshell#139>", line 1, in <module>
    name[0] = 'Д'
TypeError: 'str' object does not support item assignment
```

Вместо этого вам придется использовать комбинацию строковых функций вроде replace() или slice (которая будет рассмотрена далее):

```
>>> name = 'Маша'
>>> name.replace('М','Д')
'Даша'
>>> 'Д' + name[1:]
'Даша'
```

Извлекаем подстроки с помощью оператора [start : end : step]

Из строки можно извлечь подстроку (часть строки) с помощью функции slice. Вы определяете slice с помощью квадратных скобок, смещения начала подстроки start и конца подстроки end, а также опционального размера шага step. Некоторые из этих параметров могут быть исключены. В подстроку будут включены символы, расположенные начиная с точки, на которую указывает смещение start, и заканчивая точкой, на которую указывает смещение end.

- Оператор [:] извлекает всю последовательность от начала до конца.
- Оператор [start :] извлекает последовательность с точки, на которую указывает смещение start, до конца.
- Оператор [: end] извлекает последовательность от начала до точки, на которую указывает смещение end минус 1.
- Оператор [start : end] извлекает последовательность с точки, на которую указывает смещение start, до точки, на которую указывает смещение end минус 1.
- Оператор [start : end : step] извлекает последовательность с точки, на которую указывает смещение start, до точки, на которую указывает смещение end минус 1, опуская символы, чье смещение внутри подстроки кратно step.

Как и ранее, смещение слева направо определяется как 0, 1 и т. д., а справа налево — как -1, -2 и т. д. Если вы не укажете смещение start, функция будет использовать в качестве его значения 0 (начало строки). Если вы не укажете смещение end, функция будет использовать конец строки.

Создадим строку, содержащую русские буквы в нижнем регистре:

```
>>> letters = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
```

Использование простого двоеточия аналогично использованию последовательности 0: (целая строка):

```
>>> letters[:]
'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
```

Вот так можно получить все символы, начиная с 20-го и заканчивая последним:

```
>>> letters[20:]
'уфхцчшщъыьэюя'
```

А теперь получим символы с 12-го по 14-й (Python не включает символ, расположенный под номером, который указан последним):

```
>>> letters[12:15]
'лмн'
```

Последние три символа:

```
>>> letters[-3:]
'эюя'
```

В следующем примере мы начинаем со смещения 18 и идем до четвертого с конца символа. Обратите внимание на разницу с предыдущим примером, где старт с позиции -3 получал символ э. В этом примере конец диапазона -3 означает, что последним будет символ по адресу -4 — ь:

```
>>> letters[18:-3]
'стуфхцчшщъыь'
```

В следующем примере мы получаем символы, начиная с шестого с конца и заканчивая третьим с конца:

```
>>> letters[-6:-2]
'ъыьэ'
```

Если вы хотите увеличить шаг, укажите его после второго двоеточия, как показано в нескольких следующих примерах.

Каждый седьмой символ с начала до конца:

```
>>> letters[::7]
'ажнфы'
```

Каждый третий символ, начиная со смещения 4 и заканчивая 19-м символом:

```
>>> letters[4:20:3]
'джймпт'
```

Каждый четвертый символ, начиная с 19-го:

```
>>> letters[19::4]
'тцъю'
```

Каждый пятый символ от начала до 20-го:

```
>>> letters[:21:5]
'аейоу'
```

Опять же значение end должно быть на единицу больше, чем реальное смещение.

И это еще не все! Если задать отрицательный шаг, Python будет двигаться в обратную сторону. В следующем примере движение начинается с конца и заканчивается в начале, ни один символ не пропущен:

```
>>> letters[-1::-1]
'яюэьыъщшчцхфутсрпонмлкйизжёедгвба'
```

Можно добиться того же результата, использовав такой пример:

```
>>> letters[::-1]
'яюэьыъщшчцхфутсрпонмлкйизжёедгвба'
```

Операция slice более мягко относится к неправильным смещениям, чем поиск по индексу. Если указать смещение меньшее, чем начало строки, оно будет обрабатываться как 0, а если указать смещение большее, чем конец строки, оно будет обработано как –1. Это показано в следующих примерах.

Начиная с –50-го символа и до конца:

```
>>> letters[-50:]
'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
```

Начиная с –51-го символа и заканчивая –50-м:

```
>>> letters[-51:-50]
''
```

Получаем длину строки с помощью функции len()

До этого момента мы использовали специальные знаки препинания вроде +, чтобы манипулировать строками. Но существует не так уж много подобных функций. Теперь мы начнем использовать некоторые встроенные функции Python: именованные фрагменты кода, которые выполняют определенные операции.

Функция len() подсчитывает символы в строке:

```
>>> len(letters)
33
>>> empty = ""
>>> len(empty)
0
```

Функция len() используется также для других структур данных, которые будут рассмотрены далее.

Разделяем строку с помощью функции split()

В отличие от функции len() некоторые функции характерны только для строк. Для того чтобы использовать строковую функцию, введите имя строки, точку, имя функции и аргументы, которые нужны функции: строка. функция(аргументы). Более подробно о функциях мы будем говорить позже.

Вы можете использовать встроенную функцию split(), чтобы разбить строку на список небольших строк, основываясь на разделителе. Со списками вы познакомитесь в следующей лабораторной работе. Список — это последовательность значений, разделенных запятыми и окруженных квадратными скобками:

```
>>> todos = 'купить молоко, помыть полы, почитать ребёнку'
>>> todos.split(',')
['купить молоко', ' помыть полы', ' почитать ребёнку']
```

В предыдущем примере строка имела имя todos, а строковая функция называлась split() и получала один аргумент ','. Если вы не укажете разделитель, функция split() будет использовать любую последовательность пробелов, а также символы новой строки и табуляцию:

```
>>> todos.split()
['купить', 'молоко,', 'помыть', 'полы,', 'почитать', 'ребёнку']
```

Если вы вызываете функцию split без аргументов, вам все равно нужно добавлять круглые скобки — именно так Python узнает, что вы вызываете функцию.

Объединяем строки с помощью функции join()

Функция join() является противоположностью функции split(): она объединяет список строк в одну строку. Вызов функции выглядит немного запутанно, поскольку сначала вы указываете строку, которая объединяет остальные, а затем — список строк для объединения: string.join(list). Для того чтобы объединить список строк lines, разделив их символами новой строки, вам нужно написать '\n'.join(lines). В следующем примере мы объединим несколько имен в список, разделенный запятыми и пробелами:

```
>>> cripto_list = ['Йети', 'Полтергейст', 'Лохнесское чудовище']
>>> cripto_string = ','.join(cripto_list)
>>> print('Они обитают в лесах Нечерноземья: ', cripto_string)
Они обитают в лесах Нечерноземья:  Йети,Полтергейст,Лохнесское чудовище
```

Прочие функции для работы со строками

Python содержит большой набор функций для работы со строками. Рассмотрим принцип работы самых распространенных из них. Объектом для тестов станет следующеестихотворение:

```
>>> poem = '''Был этот мир глубокой тьмой окутан.
Да будет свет! И вот явился Ньютон.
Но Сатана недолго ждал реванша -
Пришел Эйнштейн, и стало всё как раньше.'''
```

Для начала получим первые 13 символов (их смещения лежат в диапазоне от 0 до 12):

```
>>> poem[:13]
'Был этот мир '
```

Сколько символов содержит это стихотворение? (Пробелы и символы новой строки учитываются.)

```
>>> len(poem)
145
```

Начинается ли стихотворение с буквосочетания ***Был этот***?

```
>>> poem.startswith('Был этот')
True
```

Заканчивается ли оно буквосочетанием ***всё как***?

```
>>> poem.endswith('всё как')
False
```

Найдем первое вхождениеслова ***мир***:

```
>>> word = 'мир'
>>> poem.find(word)
9
```

Найдем последнее вхождение буквы ***и***:

```
>>> poem.rfind('и')
122
```

Сравните с первым:

```
>>> poem.find('и')
10
```

Сколько раз встречается буква ***и***?

```
>>> poem.count('и')
4
```

Являются ли все символы стихотворения буквами или цифрами?

```
>>> poem.isalnum()
False
```

Нет, в стихотворении имеются еще и знаки препинания.

Регистр и выравнивание

В этом разделе мы рассмотрим еще несколько примеров использования встроенных функций. В качестве подопытной выберем следующую строку:

```
>>> setup = 'американец, француз и русский попали на необитаемый остров...'
```

Удалим символ «.» с обоих концов строки:

```
>>> setup.strip('.')
'американец, француз и русский попали на необитаемый остров'
```

> Замечание:
>
> Поскольку строки неизменяемы, ни один из этих примеров не изменяет строку setup. Каждый пример просто берет значение переменной setup, выполняет над ним некоторое действие, а затем возвращает результат как новую строку.

Напишем первое слово с большой буквы:

```
>>> setup.capitalize()
'Американец, француз и русский попали на необитаемый остров...'
```

Напишем все слова с большой буквы:

```
>>> setup.title()
'Американец, Француз И Русский Попали На Необитаемый Остров...'
```

Запишем все слова большими буквами:

```
>>> setup.upper()
'АМЕРИКАНЕЦ, ФРАНЦУЗ И РУССКИЙ ПОПАЛИ НА НЕОБИТАЕМЫЙ ОСТРОВ...'
```

Запишем все слова маленькими буквами:

```
>>> setup.lower()
'американец, француз и русский попали на необитаемый остров...'
```

Сменим регистры букв:

```
>>> setup.swapcase()
'АМЕРИКАНЕЦ, ФРАНЦУЗ И РУССКИЙ ПОПАЛИ НА НЕОБИТАЕМЫЙ ОСТРОВ...'
```

Если бы в строке были буквы в разных регистрах, они бы поменяли свой регистр на противоположный. В данном случае все буквы выведены в верхнем регистре.

Теперь мы поработаем с функциями выравнивания. Строка выравнивается внутри заданного количества пробелов (в данном примере 30). Отцентруем строку в промежутке из 30 пробелов:

```
>>> setup = 'Заходит лошадь в бар...'
>>> setup.center(30)
'   Заходит лошадь в бар...    '
```

Выровняем ее по левому краю:

```
>>> setup.ljust(30)
'Заходит лошадь в бар...       '
```

А теперь по правому:

```
>>> setup.rjust(30)
'       Заходит лошадь в бар...'
```

О форматировании и преобразовании строк мы более подробно поговорим далее. Также далее будет затронуто использование символа % и функции format().

Заменяем символы с помощью функции replace()

Вы можете использовать функцию replace() для того, чтобы заменить одну подстроку другой. Вы передаете в эту функцию старую подстроку, новую подстроку и количество включений старой подстроки, которое нужно заменить. Если вы опустите последний аргумент, будут заменены все включения. В этом примере совпадает с заданным значением и заменяется следующая строка:

```
>>> setup.replace('лошадь', 'панда')
'Заходит панда в бар...'
```

Заменим максимум 100 включений буквы ***а***:

```
>>> setup.replace('а', 'ааа', 100)
'Заааходит лошаааадь в бааар...'
```

Если вы точно знаете, какую подстроку или подстроки хотите изменить, функция replace() станет для вас хорошим выбором.

Иногда вам нужно убедиться, что подстрока является целым словом, началом слова и т. д. В этих случаях понадобятся регулярные выражения. Их мы рассмотрим в других работах.

Содержание отчета **: выполненную работу сохранить в папке под своей фамилией и показать преподавателю**

Контрольные вопросы

1. В чём особенность строк по сравнению с другими составными типами данных в Python и со строками во многих других языках программирования?
2. Какие способы задания строк вы знаете?
3. Для чего используются два вида кавычек при задании строк?
4. Как можно соединить 2 строки в одну?
5. Что будет выведено после применения команды
   ```
   >>> str(1.0e4)
   ```
6. Как извлечь из строки с русским алфавитом каждую вторую букву, начиная с «д»?
7. Какой командой можно вывести строку ‘Утро красит нежным цветом’ в обратном порядке?

8. Что будет выведено, если запросить срез переменной с русским алфавитом (33 буквы) следующим образом:
```
>>> letters[0:100]
```
9. Для чего предназначена функция len()?
10. Для чего предназначена функция split()?
11. Что будет результатом выполнения команд ?
```
>>> todos = 'купить молоко, помыть полы, почитать ребёнку'
>>> todos.split(',')
```
12. Что будет, если в предыдущем примере вызвать функцию split() без аргумента?
```
>>> todos.split()
```
13. Для чего предназначена функция join()? Приведите пример использования.
14. Для чего предназначена функция strip()?
15. Для чего предназначена функция capitalize? Что будет, если применить её к строке ‘переходи на тёмную сторону, юзернейм, у нас печеньки!’
16. Как изменится строка после применения функции capitalize?
17. Какая функция позволяет вывести все слова в строке с прописных букв? Примените её к строке ‘переходи на тёмную сторону, юзернейм, у нас печеньки!’.
18. Для чего предназначена функция upper?
19. Создайте строку «Заходит лошадь в бар». Примените функции, которые изменяют её выравнивание. Продемонстрируйте результат преподавателю.
20. Для чего используется функция replace()? Приведите пример её использования.

Список литературы, рекомендуемый к использованию по данной теме:

1. Орлов, С. А. Программная инженерия. Технологии разработки программного обеспечения : учебник / С.А. Орлов. - 5-е изд., обновл. и доп. - СПб. : Питер, 2017. - 640 с.
2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники : [учеб. пособие] / Л.Г. Гагарина, А.А. Петров. - М. : Форум, 2016. - 368 с.
3. Шкляр, М. Ф. Основы научных исследований : учеб. пособие / М.Ф. Шкляр. - 6-е изд. - М. : Дашков и Ко, 2016. - 208 с.
4. Михеева, Е. В. Информационные технологии в профессиональной деятельности : учеб. пособие / Е.В. Михеева. - 14-е изд., стер. - М. : Академия, 2016. - 384 с.
5. Любанович Билл. Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.

Лабораторная работа 3. Сложные структуры данных: списки, кортежи

Цель работы:

Изучить основные функции для работы со списками и кортежами в языке Python, особенности и области применения этих конструкций.

Компетенции:

Код	Формулировка:
ОК-5	быть готовым работать с информацией в различных формах, использовать для ее получения, обработки, передачи, хранения