Лабораторная работа 2.3 Работа со строками в языке Python

Цель работы: *приобретение навыков по работе со строками при написании программ с помощью языка программирования Python версии 3.x.*

Ход работы

Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Литералы строк

Работа со строками в Python очень удобна. Существует несколько литералов строк, которые мы сейчас и рассмотрим.

Строки в апострофах и в кавычках

```
S = 'spam"s'
S = "spam's"
```

Строки в апострофах и в кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в литералы строк символы кавычек или апострофов, не используя экранирование.

Экранированные последовательности - служебные символы

Экранированные последовательности позволяют вставить символы, которые сложно ввести с клавиатуры.

Экранированная последовательность	Назначение
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\N{id}	Идентификатор ID базы данных Юникода
\uhhhh	16-битовый символ Юникода в 16-ричном представлении
\Uhhhh...	32-битовый символ Юникода в 32-ричном представлении
\xhh	16-ричное значение символа
\ooo	8-ричное значение символа
\0	Символ Null (не является признаком конца строки)

"Сырые" строки - подавляют экранирование

Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается.

```
s = r'C:\newt.txt'
```

Но, несмотря на назначение, "сырая" строка не может заканчиваться символом обратного слэша. Пути решения:

```
s = r'\n\n\\'[:-1]
s = r'\n\n' + '\\'
s = '\\n\\n'
```

Строки в тройных апострофах или кавычках

Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

```
>>> c = '''это очень большая
... строка, многострочный
... блок текста'''
>>> c
'это очень большая\nстрока, многострочный\nблок текста'
>>> print(c)
это очень большая
строка, многострочный
блок текста
```

Строковые операторы

Вы уже видели операторы `+` и `*` в применении их к числовым значениям в предыдущих лабораторных работах . Эти два оператора применяются и к строкам.

Оператор сложения строк `+`

`+` — оператор конкатенации строк. Он возвращает строку, состоящую из других строк, как показано здесь:

```
>>> s = 'py'
>>> t = 'th'
>>> u = 'on'

>>> s + t
'pyth'
>>> s + t + u
'python'

>>> print('Привет, ' + 'Мир!')
Go team!!!
```

Оператор умножения строк `*`

`*` — оператор создает несколько копий строки. Если `s` это строка, а `n` целое число, любое из следующих выражений возвращает строку, состоящую из `n` объединенных копий `s`:

```
s * n
n * s
```

Вот примеры умножения строк:

```
>>> s = 'py.'

>>> s * 4
'py.py.py.py.'
>>> 4 * s
'py.py.py.py.'
```

Значение множителя `n` должно быть целым положительным числом. Оно может быть нулем или отрицательным, но этом случае результатом будет пустая строка:

```
>>> 'py' * -6
''
```

Оператор принадлежности подстроки `in`

Python также предоставляет оператор принадлежности, который можно использоваться для манипуляций со строками. Оператор `in` возвращает `True`, если подстрока входит в строку, и `False`, если нет:

```
>>> s = 'Python'
>>> s in 'I love Python.'
True
>>> s in 'I love Java.'
False
```

Есть также оператор `not in`, у которого обратная логика:

```
>>> 'z' not in 'abc'
True
>>> 'z' not in 'xyz'
False
```

Встроенные функции строк в python

Python предоставляет множество функций, которые встроены в интерпретатор. Вот несколько, которые работают со строками:

Функция	Описание
chr()	Преобразует целое число в символ
ord()	Преобразует символ в целое число
len()	Возвращает длину строки
str()	Изменяет тип объекта на `string`

Более подробно о них ниже.

Функция `ord(c)` возвращает числовое значение для заданного символа.

На базовом уровне компьютеры хранят всю информацию в виде цифр. Для представления символьных данных используется схема перевода, которая содержит каждый символ с его репрезентативным номером.

Самая простая схема в повседневном использовании называется **ASCII** (https://ru.wikipedia.org/wiki/ASCII). Она охватывает латинские символы, с которыми мы чаще работает. Для этих символов `ord(c)` возвращает значение ASCII для символа `c`:

```
>>> ord('a')
97
>>> ord('#')
35
```

ASCII является прекрасным выбором, если работа осуществляется только с латиницей, но есть много других языков в мире, которые часто встречаются. Полный набор символов, которые потенциально могут быть представлены в коде, намного больше обычных латинских букв, цифр и символом.

Unicode (https://www.unicode.org/standard/WhatIsUnicode.html) — это современный стандарт, который пытается предоставить числовой код для всех возможных символов, на всех возможных языках, на каждой возможной платформе. Python 3 поддерживает Unicode, в том числе позволяет использовать символы Unicode в строках.

Функция `ord()` также возвращает числовые значения для символов Юникода:

```
>>> ord('€')
8364
>>> ord('Σ')
8721
```

Функция `chr(n)` возвращает символьное значение для данного целого числа.

`chr()` действует обратно `ord()`. Если задано числовое значение `n`, `chr(n)` возвращает строку, представляющую символ `n`:

```
>>> chr(97)
'a'
>>> chr(35)
'#'
```

`chr()` также обрабатывает символы Юникода:

```
>>> chr(8364)
'€'
>>> chr(8721)
'Σ'
```

Функция `len(s)` возвращает длину строки.

`len(s)` возвращает количество символов в строке `s`:

```
>>> s = 'Простоя строка.'
>>> len(s)
15
```

Функция `str(obj)` возвращает строковое представление объекта.

Практически любой объект в Python может быть представлен как строка. `str(obj)` возвращает строковое представление объекта `obj`:

```
>>> str(49.2)
'49.2'
>>> str(3+4j)
'(3+4j)'
>>> str(3 + 29)
'32'
>>> str('py')
'py'
```

Индексация строк

Часто в языках программирования, отдельные элементы в упорядоченном наборе данных могут быть доступны с помощью числового индекса или ключа. Этот процесс называется индексация.

В Python строки являются упорядоченными последовательностями символьных данных и могут быть проиндексированы. Доступ к отдельным символам в строке можно получить, указав имя строки, за которым следует число в квадратных скобках `[]`.

Индексация строк начинается с нуля: у первого символа индекс `0`, следующего `1` и так далее. Индекс последнего символа в python — “длина строки минус один”.

Например, схематическое представление индексов строки `'foobar'` выглядит следующим образом:

Отдельные символы доступны по индексу следующим образом:

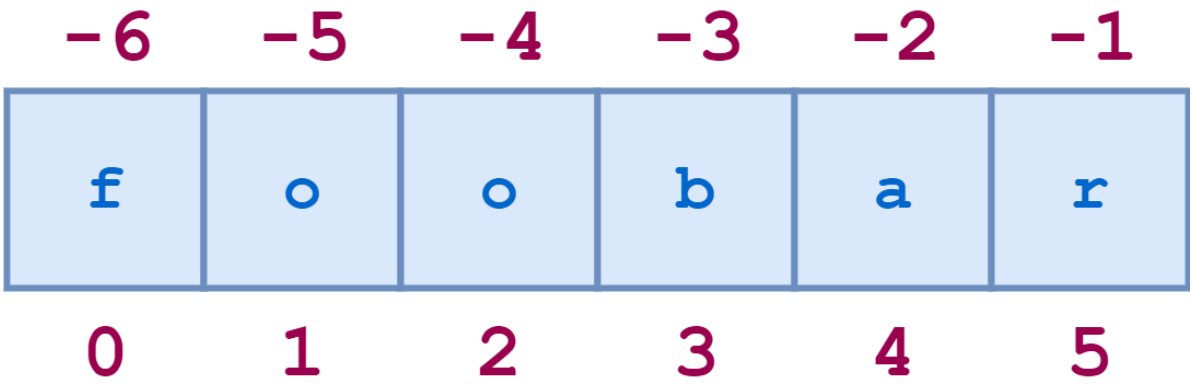
```
>>> s = 'foobar'

>>> s[0]
'f'
>>> s[1]
'o'
>>> s[3]
'b'
>>> s[5]
'r'
```

Попытка обращения по индексу большему чем `len(s) - 1`, приводит к ошибке `IndexError`:

```
>>> s[6]
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    s[6]
IndexError: string index out of range
```

Индексы строк также могут быть указаны отрицательными числами. В этом случае индексирование начинается с конца строки: `-1` относится к последнему символу, `-2` к предпоследнему и так далее. Вот такая же диаграмма, показывающая как положительные, так и отрицательные индексы строки `'foobar'`:



Вот несколько примеров отрицательного индексирования:

```
>>> s = 'foobar'
>>> s[-1]
'r'
>>> s[-2]
'a'
>>> len(s)
6
>>> s[-len(s)] # отрицательная индексация начинается с -1
'f'
```

Попытка обращения по индексу меньшему чем `-len(s)`, приводит к ошибке `IndexError`:

```
>>> s[-7]
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    s[-7]
IndexError: string index out of range
```

Для любой непустой строки `s`, код `s[len(s)-1]` и `s[-1]` возвращают последний символ. Нет индекса, который применим к пустой строке.

Срезы строк

Python также допускает возможность извлечения подстроки из строки, известную как “string slice”. Если `s` это строка, выражение формы `s[m:n]` возвращает часть `s`, начинающуюся с позиции `m`, и до позиции `n`, но не включая позицию:

```
>>> s = 'python'
>>> s[2:5]
'tho'
```

> **Помните:** индексы строк в python начинаются с нуля. Первый символ в строке имеет индекс `0`. Это относится и к срезу.

Опять же, второй индекс указывает символ, который не включен в результат. Символ `'n'` в приведенном выше примере. Это может показаться немного не интуитивным, но дает результат: выражение `s[m:n]` вернет подстроку, которая является разницей `n - m`, в данном случае `5 - 2 = 3`.

Если пропустить первый индекс, срез начинается с начала строки. Таким образом, `s[:m]` = `s[0:m]`:

```
>>> s = 'python'
>>> s[:4]
'pyth'
>>> s[0:4]
'pyth'
```

Аналогично, если опустить второй индекс `s[n:]`, срез длится от первого индекса до конца строки. Это хорошая, лаконичная альтернатива более громоздкой `s[n:len(s)]`:

```
>>> s = 'python'
>>> s[2:]
'thon'
>>> s[2:len(s)]
'thon'
```

Для любой строки `s` и любого целого `n` числа (`0 ≤ n ≤ len(s)`), `s[:n] + s[n:]` будет `s`:

```
>>> s = 'python'
>>> s[:4] + s[4:]
'python'
>>> s[:4] + s[4:] == s
True
```

Пропуск обоих индексов возвращает исходную строку. Это не копия, это ссылка на исходную строку:

```
>>> s = 'python'
>>> t = s[:]
>>> id(s)
59598496
>>> id(t)
59598496
>>> s is t
True
```

Если первый индекс в срезе больше или равен второму индексу, Python возвращает пустую строку:

```
>>> s[2:2]
''
>>> s[4:2]
''
```

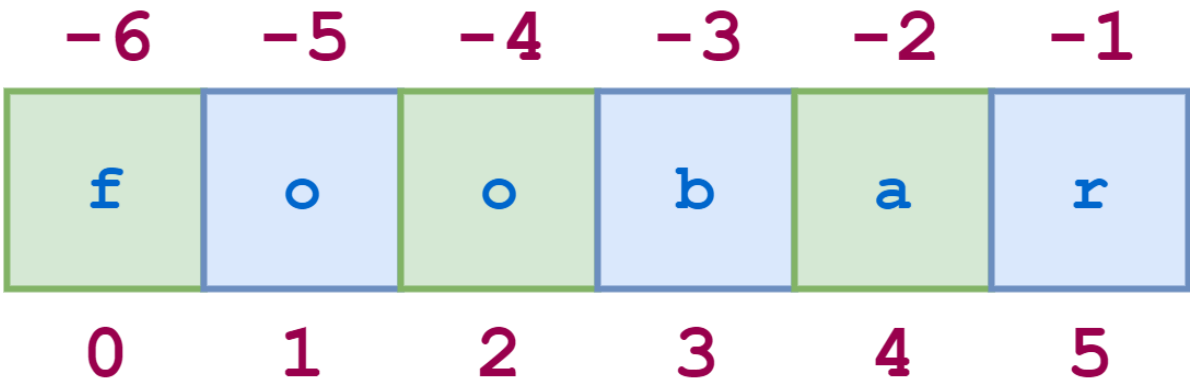
Отрицательные индексы можно использовать и со срезами. Вот пример кода Python:

```
>>> s = 'python'
>>> s[-5:-2]
'yth'
>>> s[1:4]
'yth'
>>> s[-5:-2] == s[1:4]
True
```

Шаг для среза строки

Существует еще один вариант синтаксиса среза, о котором стоит упомянуть. Добавление дополнительного `:` и третьего индекса означает шаг, который указывает, сколько символов следует пропустить после извлечения каждого символа в срезе.

Например , для строки `'python'` срез `0:6:2` начинается с первого символа и заканчивается последним символом (всей строкой), каждый второй символ пропускается. Это показано на следующей схеме:



Иллюстративный код показан здесь:

```
>>> s = 'foobar'
>>> s[0:6:2]
'foa'
>>> s[1:6:2]
'obr'
```

Как и в случае с простым срезом, первый и второй индексы могут быть пропущены:

```
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::5]
'11111'
>>> s[4::5]
'55555'
```

Вы также можете указать отрицательное значение шага, в этом случае Python идет с конца строки. Начальный/первый индекс должен быть больше конечного/второго индекса:

```
>>> s = 'python'
>>> s[5:0:-2]
'nhy'
```

В приведенном выше примере, `5:0:-2` означает «начать с последнего символа и делать два шага назад, но не включая первый символ.”

Когда вы идете назад, если первый и второй индексы пропущены, значения по умолчанию применяются так: первый индекс — конец строки, а второй индекс — начало. Вот пример:

```
>>> s = '12345' * 5
>>> s
'1234512345123451234512345'
>>> s[::-5]
'55555'
```

Это общая парадигма для разворота (reverse) строки:

```
>>> s = 'Если так говорит товарищ Наполеон, значит, так оно и есть.'
>>> s[::-1]
'.ьтсе и оно кат ,тичанз ,ноелопаН щиравот тировог кат илсЕ'
```

Форматирование строки

Иногда (а точнее, довольно часто) возникают ситуации, когда нужно сделать строку, подставив в неё некоторые данные, полученные в процессе выполнения программы (пользовательский ввод, данные из файлов и т. д.). Подстановку данных можно сделать с помощью форматирования строк. Форматирование можно сделать с помощью оператора `%`, либо с помощью метода `format`, либо с помощью так называемых f-строк. Форматирование с помощью оператора `%` относится к устаревшим способам форматирования, поэтому в рамках данной лабораторной работы будут рассмотрены только метод `format` и f-строки.

Форматирование строк с помощью метода `format`

Если для подстановки требуется только один аргумент, то значение - сам аргумент:

```
>>> 'Hello, {}!'.format('Vasya')
'Hello, Vasya!'
```

А если несколько, то значениями будут являться все аргументы со строками подстановки (обычных или именованных):

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N',
longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
'Coordinates: 37.24N, -115.81W'
```

Однако метод `format` умеет большее. Вот его синтаксис:

```
поле замены     ::=  "{" [имя поля] ["!" преобразование] [":" спецификация] "}"
имя поля        ::=  arg_name ("." имя атрибута | "[" индекс "]")*
преобразование  ::=  "r" (внутреннее представление) | "s" (человеческое
представление)
спецификация    ::=  см. ниже
```

Например:

```
>>> "Units destroyed: {players[0]}".format(players = [1, 2, 3])
'Units destroyed: 1'
>>> "Units destroyed: {players[0]!r}".format(players = ['1', '2', '3'])
"Units destroyed: '1'"
```

Теперь спецификация формата:

```
спецификация ::=  [[fill]align][sign][#][0][width][,][.precision][type]
заполнитель  ::=  символ кроме '{' или '}'
выравнивание ::=  "<" | ">" | "=" | "^"
знак         ::=  "+" | "-" | " "
ширина       ::=  integer
точность     ::=  integer
тип          ::=  "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" |
                  "n" | "o" | "s" | "x" | "X" | "%"
```

Выравнивание производится при помощи символа-заполнителя. Доступны следующие варианты выравнивания:

Флаг	Значение
'<'	Символы-заполнители будут справа (выравнивание объекта по левому краю) (по умолчанию).
'>'	выравнивание объекта по правому краю.
'='	Заполнитель будет после знака, но перед цифрами. Работает только с числовыми типами.
'^'	Выравнивание по центру.

Опция "знак" используется только для чисел и может принимать следующие значения:

Флаг	Значение
'+'	Знак должен быть использован для всех чисел.
'-'	'-' для отрицательных, ничего для положительных.
'Пробел'	'-' для отрицательных, пробел для положительных.

Поле "тип" может принимать следующие значения:

Тип	Значение
'd', 'i', 'u'	Десятичное число.
'o'	Число в восьмеричной системе счисления.
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'f', 'F'	Число с плавающей точкой (обычный формат).
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'c'	Символ (строка из одного символа или число - код символа).
's'	Строка.
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %.

И напоследок, несколько примеров:

```
>>> coord = (3, 5)
>>> 'X: {0[0]};  Y: {0[1]}'.format(coord)
'X: 3;  Y: 5'
>>> "repr() shows quotes: {!r}; str() doesn't: {!s}".format('test1', 'test2')
"repr() shows quotes: 'test1'; str() doesn't: test2"
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill char
'***********centered***********'
>>> '{:+f}; {:+f}'.format(3.14, -3.14)  # show it always
'+3.140000; -3.140000'
>>> '{: f}; {: f}'.format(3.14, -3.14)  # show a space for positive numbers
' 3.140000; -3.140000'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)  # show only the minus -- same as '{:f};
{:f}'
'3.140000; -3.140000'
>>> # format also supports binary numbers
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42)
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
>>> # with 0x, 0o, or 0b as prefix:
```

```
>>> "int: {0:d};  hex: {0:#x};  oct: {0:#o};  bin: {0:#b}".format(42)
'int: 42;  hex: 0x2a;  oct: 0o52;  bin: 0b101010'
>>> points = 19.5
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 88.64%'
```

Форматирование с момощью f-строк

В Python версии 3.6 был представлен новый способ форматирования строк. Эта функция официально названа литералом отформатированной строки, но обычно упоминается как f-строки (f-string).

Возможности форматирования строк огромны и не будут подробно описана здесь.
Одной простой особенностью f-строк, которые вы можете начать использовать сразу, является интерполяция переменной. Вы можете указать имя переменной непосредственно в f-строковом литерале (f'string'), и python заменит имя соответствующим значением.

Например, предположим, что вы хотите отобразить результат арифметического вычисления. Это можно сделать с помощью простого print() и оператора ,, разделяющего числовые значения и строковые:

```
>>> n = 20
>>> m = 25
>>> prod = n * m
>>> print('Произведение', n, 'на', m, 'равно', prod)
Произведение 20 на 25 равно 500
```

Но это громоздко. Чтобы выполнить то же самое с помощью f-строки:

- Напишите `f` или `F` перед кавычками строки. Это укажет python, что это f-строка вместо стандартной.
- Укажите любые переменные для воспроизведения в фигурных скобках (`{}`).

Код с использованием f-string, приведенный ниже выглядит намного чище:

```
>>> n = 20
>>> m = 25
>>> prod = n * m
>>> print(f'Произведение {n} на {m} равно {prod}')
Произведение 20 на 25 равно 500
```

Любой из трех типов кавычек в python можно использовать для f-строки:

```
>>> var = 'Гав'
>>> print(f'Собака говорит {var}!')
Собака говорит Гав!
>>> print(f"Собака говорит {var}!")
Собака говорит Гав!
>>> print(f'''Собака говорит {var}!''')
Собака говорит Гав!
```

Изменение строк

Строки — один из типов данных, которые Python считает неизменяемыми, что означает невозможность их изменять. Как вы ниже увидите, python дает возможность изменять (заменять и перезаписывать) строки.

Такой синтаксис приведет к ошибке `TypeError`:

```
>>> s = 'python'
>>> s[3] = 't'
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    s[3] = 't'
TypeError: 'str' object does not support item assignment
```

На самом деле нет особой необходимости изменять строки. Обычно вы можете легко сгенерировать копию исходной строки с необходимыми изменениями. Есть минимум 2 способа сделать это в python. Вот первый:

```
>>> s = s[:3] + 't' + s[4:]
>>> s
'pytton'
```

Есть встроенный метод `string.replace(x, y)`:

```
>>> s = 'python'
>>> s = s.replace('h', 't')
>>> s
'pytton'
```

Встроенные методы строк в python

Каждый элемент данных в программе python является объектом. Функции являются самостоятельными блоками кода, которые вы можете вызывать для выполнения определенных задач.

Методы похожи на функции. Метод — специализированный тип вызываемой процедуры, тесно связанный с объектом. Как и функция, метод вызывается для выполнения отдельной задачи, но он вызывается только вместе с определенным объектом и знает о нем во время выполнения.

Синтаксис для вызова метода объекта выглядит следующим образом:

```
obj.foo(<args>)
```

Этот код вызывает метод `.foo()` объекта `obj`. `<args>` — аргументы, передаваемые методу (если есть).

Более подробно с функциями, объектами и методыми вы познакомитесь в следующих лабораторных работах. В этой лабораторной работе интерес представляют только методы для работы со строками.

В приведенных методах аргументы, указанные в квадратных скобках (`[]`), являются необязательными.

Изменение регистра строки

Методы этой группы выполняют преобразование регистра строки.

`string.capitalize()` приводит первую букву в верхний регистр, остальные в нижний.

`s.capitalize()` возвращает копию `s` с первым символом, преобразованным в верхний регистр, и остальными символами, преобразованными в нижний регистр:

```
>>> s = 'everyTHing yoU Can IMaGine is rEAl'
>>> s.capitalize()
'Everything you can imagine is real'
```

Не алфавитные символы не изменяются:

```
>>> s = 'follow us @PYTHON'
>>> s.capitalize()
'Follow us @python'
```

`string.lower()` преобразует все буквенные символы в строчные.

`s.lower()` возвращает копию `s` со всеми буквенными символами, преобразованными в нижний регистр:

```
>>> 'everyTHing yoU Can IMaGine is rEAl'.lower()
'everything you can imagine is real'
```

`string.swapcase()` меняет регистр буквенных символов на противоположный.

`s.swapcase()` возвращает копию `s` с заглавными буквенными символами, преобразованными в строчные и наоборот:

```
>>> 'everyTHing yoU Can IMaGine is rEAl'.swapcase()
'EVERYthING YOu cAN imAgINE IS ReaL'
```

`string.title()` преобразует первые буквы всех слов в заглавные

`s.title()` возвращает копию, `s` в которой первая буква каждого слова преобразуется в верхний регистр, а остальные буквы — в нижний регистр:

```
>>> 'the sun also rises'.title()
'The Sun Also Rises'
```

Этот метод использует довольно простой алгоритм. Он не пытается различить важные и неважные слова и не обрабатывает апострофы, имена или аббревиатуры:

```
>>> 'follow us @PYTHON'.title()
'Follow Us @Python'
```

`string.upper()` преобразует все буквенные символы в заглавные.

`s.upper()` возвращает копию `s` со всеми буквенными символами в верхнем регистре:

```
>>> 'follow us @PYTHON'.upper()
'FOLLOW US @PYTHON'
```

Найти и заменить подстроку в строке

Эти методы предоставляют различные способы поиска в целевой строке указанной подстроки.

Каждый метод в этой группе поддерживает необязательные аргументы `<start>` и `<end>` аргументы. Они задают диапазон поиска: действие метода ограничено частью целевой строки, начинающейся в позиции символа `<start>` и продолжающейся вплоть до позиции символа `<end>`, но не включая его. Если `<start>` указано, а `<end>` нет, метод применяется к части строки от `<start>` конца.

`string.count(<sub>[, <start>[, <end>]])` подсчитывает количество вхождений подстроки в строку.

`s.count(<sub>)` возвращает количество точных вхождений подстроки `<sub>` в `s`:

```
>>> 'foo goo moo'.count('oo')
3
```

Количество вхождений изменится, если указать `<start>` и `<end>`:

```
>>> 'foo goo moo'.count('oo', 0, 8)
2
```

`string.endswith(<suffix>[, <start>[, <end>]])` определяет, заканчивается ли строка заданной подстрокой.

`s.endswith(<suffix>)` возвращает, `True` если `s` заканчивается указанным `<suffix>` и `False` если нет:

```
>>> 'python'.endswith('on')
True
>>> 'python'.endswith('or')
False
```

Сравнение ограничено подстрокой, между `<start>` и `<end>`, если они указаны:

```
>>> 'python'.endswith('yt', 0, 4)
True
>>> 'python'.endswith('yt', 2, 4)
False
```

`string.find(<sub>[, <start>[, <end>]])` ищет в строке заданную подстроку.

`s.find(<sub>)` возвращает первый индекс в `s` который соответствует началу строки `<sub>`:

```
>>> 'Follow Us @Python'.find('Us')
7
```

Этот метод возвращает, `-1` если указанная подстрока не найдена:

```
>>> 'Follow Us @Python'.find('you')
-1
```

Поиск в строке ограничивается подстрокой, между `<start>` и `<end>`, если они указаны:

```
>>> 'Follow Us @Python'.find('Us', 4)
7
>>> 'Follow Us @Python'.find('Us', 4, 7)
-1
```

`string.index(<sub>[, <start>[, <end>]])` ищет в строке заданную подстроку.

Этот метод идентичен `.find()`, за исключением того, что он вызывает исключение `ValueError`, если `<sub>` не найден:

```
>>> 'Follow Us @Python'.index('you')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    'Follow Us @Python'.index('you')
ValueError: substring not found
```

`string.rfind(<sub>[, <start>[, <end>]])` ищет в строке заданную подстроку, начиная с конца.

`s.rfind(<sub>)` возвращает индекс последнего вхождения подстроки `<sub>` в `s`, который соответствует началу `<sub>`:

```
>>> 'Follow Us @Python'.rfind('o')
15
```

Как и в `.find()`, если подстрока не найдена, возвращается `-1`:

```
>>> 'Follow Us @Python'.rfind('a')
-1
```

Поиск в строке ограничивается подстрокой, между `<start>` и `<end>`, если они указаны:

```
>>> 'Follow Us @Python'.rfind('Us', 0, 14)
7
>>> 'Follow Us @Python'.rfind('Us', 9, 14)
-1
```

`string.rindex(<sub>[, <start>[, <end>]])` ищет в строке заданную подстроку, начиная с конца.

Этот метод идентичен `.rfind()`, за исключением того, что он вызывает исключение `ValueError`, если `<sub>` не найден:

```
>>> 'Follow Us @Python'.rindex('you')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    'Follow Us @Python'.rindex('you')
ValueError: substring not found
```

`string.startswith(<prefix>[, <start>[, <end>]])` определяет, начинается ли строка с заданной подстроки.

`s.startswith(<suffix>)` возвращает, `True` если `s` начинается с указанного `<suffix>` и `False` если нет:

```
>>> 'Follow Us @Python'.startswith('Fol')
True
>>> 'Follow Us @Python'.startswith('Go')
False
```

Сравнение ограничено подстрокой, между `<start>` и `<end>`, если они указаны:

```
>>> 'Follow Us @Python'.startswith('Us', 7)
True
>>> 'Follow Us @Python'.startswith('Us', 8, 16)
False
```

Классификация строк

Методы в этой группе классифицируют строку на основе символов, которые она содержит.

`string.isalnum()` определяет, состоит ли строка из букв и цифр.

`s.isalnum()` возвращает `True`, если строка `s` не пустая, а все ее символы буквенно-цифровые (либо буква, либо цифра). В другом случае `False` :

```
>>> 'abc123'.isalnum()
True
>>> 'abc$123'.isalnum()
False
>>> ''.isalnum()
False
```

`string.isalpha()` определяет, состоит ли строка только из букв.

`s.isalpha()` возвращает `True`, если строка `s` не пустая, а все ее символы буквенные. В другом случае `False` :

```
>>> 'ABCabc'.isalpha()
True
>>> 'abc123'.isalpha()
False
```

`string.isdigit()` определяет, состоит ли строка из цифр (проверка на число).

`s.digit()` возвращает `True` когда строка `s` не пустая и все ее символы являются цифрами, а в `False` если нет:

```
>>> '123'.isdigit()
True
>>> '123abc'.isdigit()
False
```

`string.isidentifier()` определяет, является ли строка допустимым идентификатором Python.

`s.isidentifier()` возвращает `True`, если `s` валидный идентификатор (название переменной, функции, класса и т.д.) python, а в `False` если нет:

```
>>> 'foo32'.isidentifier()
True
>>> '32foo'.isidentifier()
False
>>> 'foo$32'.isidentifier()
False
```

Важно: `.isidentifier()` вернет `True` для строки, которая соответствует зарезервированному ключевому слову python, даже если его нельзя использовать:

```
>>> 'and'.isidentifier()
True
```

Вы можете проверить, является ли строка ключевым словом Python, используя функцию `iskeyword()`, которая находится в модуле `keyword`. Один из возможных способов сделать это:

```
>>> from keyword import iskeyword
>>> iskeyword('and')
True
```

Если вы действительно хотите убедиться, что строку можно использовать как идентификатор python, вы должны проверить, что `.isidentifier()` = `True` и `iskeyword()` = `False`.

`string.islower()` определяет, являются ли буквенные символы строки строчными.

`s.islower()` возвращает `True`, если строка `s` не пустая, и все содержащиеся в нем буквенные символы строчные, а `False` если нет. Не алфавитные символы игнорируются:

```
>>> 'abc'.islower()
True
>>> 'abc1$d'.islower()
True
>>> 'Abc1$D'.islower()
False
```

`string.isprintable()` определяет, состоит ли строка только из печатаемых символов.

`s.isprintable()` возвращает, `True` если строка `s` пустая или все буквенные символы которые она содержит можно вывести на экран. Возвращает, `False` если `s` содержит хотя бы один специальный символ. Не алфавитные символы игнорируются:

```
>>> 'a\tb'.isprintable() # \t - символ табуляции
False
>>> 'a b'.isprintable()
True
>>> ''.isprintable()
True
>>> 'a\nb'.isprintable() # \n - символ перевода строки
False
```

> **Важно:** Это единственный `.is****()` метод, который возвращает `True`, если `s` пустая строка. Все остальные возвращаются `False`.

`string.isspace()` определяет, состоит ли строка только из пробельных символов.

`s.isspace()` возвращает `True`, если `s` не пустая строка, и все символы являются пробельными, а `False`, если нет.

Наиболее часто встречающиеся пробельные символы — это пробел `' '`, табуляция `'\t'` и новая строка `'\n'`:

```
>>> ' \t  \n '.isspace()
True
>>> ' a '.isspace()
False
```

Тем не менее есть несколько символов ASCII, которые считаются пробелами. И если учитывать символы Юникода, их еще больше:

```
>>> '\f\u2005\r'.isspace()
True
```

`'\f'` и `'\r'` являются escape-последовательностями для символов ASCII; `'\u2005'` это escape-последовательность для Unicode.

`string.istitle()` определяет, начинаются ли слова строки с заглавной буквы.

`s.istitle()` возвращает `True` когда `s` не пустая строка и первый алфавитный символ каждого слова в верхнем регистре, а все остальные буквенные символы в каждом слове строчные. Возвращает `False`, если нет:

```
>>> 'This Is A Title'.istitle()
True
>>> 'This is a title'.istitle()
False
>>> 'Give Me The #$#@ Ball!'.istitle()
True
```

`string.isupper()` определяет, являются ли буквенные символы строки заглавными.

`s.isupper()` возвращает `True`, если строка `s` не пустая, и все содержащиеся в ней буквенные символы являются заглавными, и в `False`, если нет. Не алфавитные символы игнорируются:

```
>>> 'ABC'.isupper()
True
>>> 'ABC1$D'.isupper()
True
>>> 'Abc1$D'.isupper()
False
```

Выравнивание строк, отступы

Методы в этой группе влияют на вывод строки.

`string.center(<width>[, <fill>])` выравнивает строку по центру.

`s.center(<width>)` возвращает строку, состоящую из `s` выровненной по ширине `<width>`. По умолчанию отступ состоит из пробела ASCII:

```
>>> 'py'.center(10)
'    py    '
```

Если указан необязательный аргумент `<fill>`, он используется как символ заполнения:

```
>>> 'py'.center(10, '-')
'----py----'
```

Если `s` больше или равна `<width>`, строка возвращается без изменений:

```
>>> 'python'.center(2)
'python'
```

`string.expandtabs(tabsize=8)` заменяет табуляции на пробелы

`s.expandtabs()` заменяет каждый символ табуляции (`'\t'`) пробелами. По умолчанию табуляция заменяются на 8 пробелов:

```
>>> 'a\tb\tc'.expandtabs()
'a       b       c'
>>> 'aaa\tbbb\tc'.expandtabs()
'aaa     bbb     c'
```

`tabsize` необязательный параметр, задающий количество пробелов:

```
>>> 'a\tb\tc'.expandtabs(4)
'a   b   c'
>>> 'aaa\tbbb\tc'.expandtabs(tabsize=4)
'aaa bbb c'
```

`string.ljust(<width>[, <fill>])` выравнивание по левому краю строки в поле.

`s.ljust(<width>)` возвращает строку `s`, выравненную по левому краю в поле шириной `<width>`. По умолчанию отступ состоит из пробела ASCII:

```
>>> 'python'.ljust(10)
'python    '
```

Если указан аргумент `<fill>` , он используется как символ заполнения:

```
>>> 'python'.ljust(10, '-')
'python----'
```

Если `s` больше или равна `<width>`, строка возвращается без изменений:

```
>>> 'python'.ljust(2)
'python'
```

`string.lstrip([<chars>])` обрезает пробельные символы слева

`s.lstrip()` возвращает копию `s` в которой все пробельные символы с левого края удалены:

```
>>> '   foo bar baz   '.lstrip()
'foo bar baz   '
>>> '\t\nfoo\t\nbar\t\nbaz'.lstrip()
'foo\t\nbar\t\nbaz'
```

Необязательный аргумент `<chars>`, определяет набор символов, которые будут удалены:

```
>>> 'https://www.pythonru.com'.lstrip('/:pths')
'www.pythonru.com'
```

`string.replace(<old>, <new>[, <count>])` заменяет вхождения подстроки в строке.

`s.replace(<old>, <new>)` возвращает копию `s` где все вхождения подстроки `<old>`, заменены на `<new>`:

```
>>> 'I hate python! I hate python! I hate python!'.replace('hate', 'love')
'I love python! I love python! I love python!'
```

Если указан необязательный аргумент `<count>`, выполняется количество `<count>` замен:

```
>>> 'I hate python! I hate python! I hate python!'.replace('hate', 'love', 2)
'I love python! I love python! I hate python!'
```

`string.rjust(<width>[, <fill>])` выравнивание по правому краю строки в поле.

`s.rjust(<width>)` возвращает строку `s`, выравненную по правому краю в поле шириной `<width>`. По умолчанию отступ состоит из пробела ASCII:

```
>>> 'python'.rjust(10)
'    python'
```

Если указан аргумент `<fill>` , он используется как символ заполнения:

```
>>> 'python'.rjust(10, '-')
'----python'
```

Если `s` больше или равна `<width>`, строка возвращается без изменений:

```
>>> 'python'.rjust(2)
'python'
```

`string.rstrip([<chars>])` обрезает пробельные символы справа

`s.rstrip()` возвращает копию `s` без пробельных символов, удаленных с правого края:

```
>>> '   foo bar baz   '.rstrip()
'   foo bar baz'
>>> 'foo\t\nbar\t\nbaz\t\n'.rstrip()
'foo\t\nbar\t\nbaz'
```

Необязательный аргумент `<chars>`, определяет набор символов, которые будут удалены:

```
>>> 'foo.$$$;'.rstrip(';$.')
'foo'
```

`string.strip([<chars>])` удаляет символы с левого и правого края строки.

`s.strip()` эквивалентно последовательному вызову `s.lstrip()` и `s.rstrip()`. Без аргумента `<chars>` метод удаляет пробелы в начале и в конце:

```
>>> s = '   foo bar baz\t\t\t'
>>> s = s.lstrip()
>>> s = s.rstrip()
>>> s
'foo bar baz'
```

Как в `.lstrip()` и `.rstrip()`, необязательный аргумент `<chars>` определяет набор символов, которые будут удалены:

```
>>> 'www.pythonru.com'.strip('w.moc')
'pythonru'
```

> **Важно:** Когда возвращаемое значение метода является другой строкой, как это часто бывает, методы можно вызывать последовательно:

```
>>> '   foo bar baz\t\t\t'.lstrip().rstrip()
'foo bar baz'
>>> '   foo bar baz\t\t\t'.strip()
'foo bar baz'

>>> 'www.pythonru.com'.lstrip('w.').rstrip('.moc')
'pythonru'
>>> 'www.pythonru.com'.strip('w.moc')
'pythonru'
```

`string.zfill(<width>)` дополняет строку нулями слева.

`s.zfill(<width>)` возвращает копию `s` дополненную `'0'` слева для достижения длины строки указанной в `<width>`:

```
>>> '42'.zfill(5)
'00042'
```

Если `s` содержит знак перед цифрами, он остается слева строки:

```
>>> '+42'.zfill(8)
'+0000042'
>>> '-42'.zfill(8)
'-0000042'
```

Если `s` больше или равна `<width>`, строка возвращается без изменений:

```
>>> '-42'.zfill(3)
'-42'
```

`.zfill()` наиболее полезен для строковых представлений чисел, но python с удовольствием заполнит строку нулями, даже если в ней нет чисел:

```
>>> 'foo'.zfill(6)
'000foo'
```

Методы преобразование строки в список

Методы в этой группе преобразовывают строку в другой тип данных и наоборот. Эти методы возвращают или принимают итерируемые объекты — термин Python для последовательного набора объектов.

Многие из этих методов возвращают либо список, либо кортеж. Это два похожих типа данных, которые являются прототипами примеров итераций в python. Список заключен в квадратные скобки (`[]`), а кортеж заключен в простые (`()`).

Теперь давайте посмотрим на последнюю группу строковых методов.

`string.join(<iterable>)` объединяет список в строку.

`s.join(<iterable>)` возвращает строку, которая является результатом конкатенации объекта `<iterable>` с разделителем `s`.

Обратите внимание, что `.join()` вызывается строка-разделитель `s` . `<iterable>` должна быть последовательностью строковых объектов.

Примеры кода помогут вникнуть. В первом примере разделителем `s` является строка `', '`, а `<iterable>` список строк:

```
>>> ', '.join(['foo', 'bar', 'baz', 'qux'])
'foo, bar, baz, qux'
```

В результате получается одна строка, состоящая из списка объектов, разделенных запятыми.

В следующем примере `<iterable>` указывается как одно строковое значение. Когда строковое значение используется в качестве итерируемого, оно интерпретируется как список отдельных символов строки:

```
>>> list('corge')
['c', 'o', 'r', 'g', 'e']

>>> ':'.join('corge')
'c:o:r:g:e'
```

Таким образом, результатом `':'.join('corge')` является строка, состоящая из каждого символа в `'corge'`, разделенного символом `':'`.

Этот пример завершается с ошибкой `TypeError`, потому что один из объектов в `<iterable>` не является строкой:

```
>>> '---'.join(['foo', 23, 'bar'])
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    '---'.join(['foo', 23, 'bar'])
TypeError: sequence item 1: expected str instance, int found
```

Это можно исправить так:

```
>>> '---'.join(['foo', str(23), 'bar'])
'foo---23---bar'
```

Как вы скоро увидите, многие объекты в Python можно итерировать, и `.join()` особенно полезен для создания из них строк.

`string.partition(<sep>)` делит строку на основе разделителя.

`s.partition(<sep>)` отделяет от `s` подстроку длиной от начала до первого вхождения `<sep>`. Возвращаемое значение представляет собой кортеж из трех частей:

- Часть `s` до `<sep>`
- Разделитель `<sep>`
- Часть `s` после `<sep>`

Вот пара примеров `.partition()` в работе:

```
>>> 'foo.bar'.partition('.')
('foo', '.', 'bar')
>>> 'foo@@bar@@baz'.partition('@@')
('foo', '@@', 'bar@@baz')
```

Если `<sep>` не найден в `s`, возвращаемый кортеж содержит `s` и две пустые строки:

```
>>> 'foo.bar'.partition('@@')
('foo.bar', '', '')
```

`s.rpartition(<sep>)` делит строку на основе разделителя, начиная с конца.

`s.rpartition(<sep>)` работает как `s.partition(<sep>)`, за исключением того, что `s` делится при последнем вхождении `<sep>` вместо первого:

```
>>> 'foo@@bar@@baz'.partition('@@')
('foo', '@@', 'bar@@baz')

>>> 'foo@@bar@@baz'.rpartition('@@')
('foo@@bar', '@@', 'baz')
```

`string.rsplit(sep=None, maxsplit=-1)` делит строку на список из подстрок.

Без аргументов `s.rsplit()` делит `s` на подстроки, разделенные любой последовательностью пробелов, и возвращает список:

```
>>> 'foo bar baz qux'.rsplit()
['foo', 'bar', 'baz', 'qux']
>>> 'foo\n\tbar   baz\r\fqux'.rsplit()
['foo', 'bar', 'baz', 'qux']
```

Если `<sep>` указан, он используется в качестве разделителя:

```
>>> 'foo.bar.baz.qux'.rsplit(sep='.')
['foo', 'bar', 'baz', 'qux']
```

Если `<sep>` = `None`, строка разделяется пробелами, как если бы `<sep>` не был указан вообще.

Когда `<sep>` явно указан в качестве разделителя `s`, последовательные повторы разделителя будут возвращены как пустые строки:

```
>>> 'foo...bar'.rsplit(sep='.')
['foo', '', '', 'bar']
```

Это не работает, когда `<sep>` не указан. В этом случае последовательные пробельные символы объединяются в один разделитель, и результирующий список никогда не будет содержать пустых строк:

```
>>> 'foo\t\t\tbar'.rsplit()
['foo', 'bar']
```

Если указан необязательный параметр `<maxsplit>`, выполняется максимальное количество разделений, начиная с правого края `s`:

```
>>> 'www.pythonru.com'.rsplit(sep='.', maxsplit=1)
['www.pythonru', 'com']
```

Значение по умолчанию для `<maxsplit>` — `-1`. Это значит, что все возможные разделения должны быть выполнены:

```
>>> 'www.pythonru.com'.rsplit(sep='.', maxsplit=-1)
['www', 'pythonru', 'com']
>>> 'www.pythonru.com'.rsplit(sep='.')
['www', 'pythonru', 'com']
```

`string.split(sep=None, maxsplit=-1)` делит строку на список из подстрок.

`s.split()` ведет себя как `s.rsplit()`, за исключением того, что при указании `<maxsplit>`, деление начинается с левого края `s`:

```
>>> 'www.pythonru.com'.split('.', maxsplit=1)
['www', 'pythonru.com']
>>> 'www.pythonru.com'.rsplit('.', maxsplit=1)
['www.pythonru', 'com']
```

Если `<maxsplit>` не указано, между `.rsplit()` и `.split()` в python разницы нет.

`string.splitlines([<keepends>])` делит текст на список строк.

`s.splitlines()` делит `s` на строки и возвращает их в списке. Любой из следующих символов или последовательностей символов считается границей строки:

Разделитель	**Значение**
\n	Новая строка
\r	Возврат каретки
\r\n	Возврат каретки + перевод строки
\v или же \x0b	Таблицы строк
\f или же \x0c	Подача формы
\x1c	Разделитель файлов
\x1d	Разделитель групп
\x1e	Разделитель записей
\x85	Следующая строка
\u2028	Новая строка (Unicode)
\u2029	Новый абзац (Unicode)

Вот пример использования нескольких различных разделителей строк:

```
>>> 'foo\nbar\r\nbaz\fqux\u2028quux'.splitlines()
['foo', 'bar', 'baz', 'qux', 'quux']
```

Если в строке присутствуют последовательные символы границы строки, они появятся в списке результатов, как пустые строки:

```
>>> 'foo\f\f\fbar'.splitlines()
['foo', '', '', 'bar']
```

Если необязательный аргумент `<keepends>` указан и его булевое значение `True`, то символы границы строк сохраняются в списке подстрок:

```
>>> 'foo\nbar\nbaz\nqux'.splitlines(True)
['foo\n', 'bar\n', 'baz\n', 'qux']
>>\> 'foo\nbar\nbaz\nqux'.splitlines(8)
['foo\n', 'bar\n', 'baz\n', 'qux']
```

Пример 1. *Дано предложение. Все пробелы в нем заменить символом «_».*

Решение: Напишем программу для решения поставленной задачи.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

if __name__ == '__main__':
    s = input("Введите предложение: ")
    r = s.replace(' ', '_')
    print("Предложение после замены: {r}")
```

В данном примере для замены символов пробела на символ подчеркивания используется метод `replace`.

Пример 2. *Дано слово. Если его длина нечетная, то удалить среднюю букву, в противном случае – две средние буквы.*

Решение: Напишем программу для решения поставленной задачи.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

if __name__ == '__main__':
    word = input("Введите слово: ")

    idx = len(word) // 2
    if len(word) % 2 == 1:
        # Длина слова нечетная.
        r = word[:idx] + word[idx+1:]
    else:
        # Длина слова четная.
        r = word[:idx-1] + word[idx+1:]

    print(r)
```

В данном примере для удаления символов используются срезы строк с последующей их конкатенацией.

Пример 3. *Дана строка текста, в котором нет начальных и конечных пробелов. Необходимо изменить ее так, чтобы длина строки стала равна заданной длине (предполагается, что требуемая длина не меньше исходной). Это следует сделать путем вставки между словами дополнительных пробелов. Количество пробелов между отдельными словами должно отличаться не более чем на 1.*

Решение: Напишем программу для решения поставленной задачи.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```python
import sys


if __name__ == '__main__':
    s = input("Введите предложение: ")
    n = int(input("Введите длину: "))

    # Проверить требуемую длину.
    if len(s) >= n:
        print(
            "Заданная длина должна быть больше длины предложения",
            file=sys.stderr
        )
        exit(1)

    # Разделить предложение на слова.
    words = s.split(' ')
    # Проверить количество слов в предложении.
    if len(words) < 2:
        print(
            "Предложение должно содержать несколько слов",
            file=sys.stderr
        )
        exit(1)

    # Количество пробелов для добавления.
    delta = n
    for word in words:
        delta -= len(word)

    # Количество пробелов на каждое слово.
    w, r = delta // (len(words) - 1), delta % (len(words) - 1)

    # Сформировать список для хранения слов и пробелов.
    lst = []

    # Пронумеровать все слова в списке и перебрать их.
    for i, word in enumerate(words):
        lst.append(word)

        # Если слово не является последним, добавить пробелы.
        if i < len(words) - 1:
            # Определить количество пробелов.
            width = w
            if r > 0:
                width += 1
                r -= 1

            # Добавить заданное количество пробелов в список.
            if width > 0:
                lst.append(' ' * width)

    # Вывести новое предложение, объединив все элементы списка lst.
    print(''.join(lst))
```

В данном примере для решения поставленной задачи сначала предложение разбивается на слова посредством вызова метода `split`, разделителем выступает символ пробела. В результате создается список слов `words` (со списками и кортежами мы познакомимся в следующей лабораторной работе). Затем определяется число пробелов `delta`, которое требуется добавить к словам в предложении. Величина `delta` должна быть распределена равномерно между всеми словами предложения. Поскольку пробелы должны быть добавлены между словами в предложении, то количество пробелов равно количеству слов минус единица (пробелов после последнего слова не должно быть).

Далее создается новый пустой список `lst` для хранения слов и пробелов между ними. После чего осуществляется перебор элементов списка `words` вместе с их индексами с использованием вспомогательной функции `enumerate`. Можно было бы не использовать эту функцию, а организовать цикл следующим образом:

```
for i in range(words):
    lst.append(words[i])
    . . .
```

Однако этот способ не соответствует идеологии программирования на языке Python (так называемому pythonic way) и чреват появлением ошибок в коде.

В теле цикла к списку `lst` добавляется слово из списка `words` посредством вызова метода `append`. Если слово не является последним в списке `words`, то к нему также добавляется необходимое число пробелов.

После завершения цикла, элементы списка `lst` объединяются с использованием метода `join`. Результат объединения выводится на экран.

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub, рекомендован к использованию Google Chrome.
5. Дистрибутив языка программирования Python, включающий набор популярных библиотек Anaconda.
6. Интегрированная среда разработки PyCharm Community Edition.

Указания по технике безопасности

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;
- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.

2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл `.gitignore` необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы. Создайте для каждого примера отдельный модуль языка Python. Зафиксируйте изменения в репозитории.
8. Приведите в отчете скриншоты результатов выполнения каждой из программ примеров при различных исходных данных вводимых с клавиатуры.
9. Выполните индивидуальные задания, согласно своего варианта. Для заданий повышенной сложности номер варианта должен быть получен у преподавателя.
10. Зафиксируйте сделанные изменения в репозитории.
11. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.
12. Выполните слияние ветки для разработки с веткой *main* / *master*.
13. Отправьте сделанные изменения на сервер GitHub.
14. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Индивидуальные задания

Задание 1

Решить задачу согласно варианта, составить программу на языке программирования Python. Номер варианта необходимо получить у преподавателя.

1. Дано название футбольного клуба. Напечатать его на экране «столбиком».
2. Составить программу, которая печатает заданное слово, начиная с последней буквы.
3. Дано слово s_1. Получить слово s_2, образованное нечетными буквами слова s_1.
4. Дано слово. Добавить к нему в начале четыре символа «+» и в конце – пять символов «–».
5. Дано слово. Добавить к нему в начале и конце столько звездочек, сколько букв в этом слове.
6. Даны два слова (первое длиннее второго). Заменить во втором слове соответствующее количество символов на первое слово.
7. Дано предложение. Напечатать все его буквы и.
8. Дано предложение. Составить программу, которая печатает «столбиком» все вхождения в предложение некоторого символа.
9. Дано предложение. Вывести «столбиком» его третий, шестой и т. д. символы.
10. Дано предложение. Вывести все буквы м и н в нем.
11. Дано предложение. Составить программу, которая выводит все вхождения в предложение двух заданных символов.
12. Дано предложение. Вывести все имеющиеся в нем буквосочетания нн.
13. Дано предложение. Вывести «столбиком» все его буквы и, стоящие на четных местах.
14. Дано предложение. Вывести «столбиком» его первый, второй, пятый, шестой, девятый, десятый и т. д. символы.
15. Дано предложение. Определить число букв о в нем.
16. Дано предложение. Определить число пробелов в нем.
17. Дано предложение. Определить число вхождений в него некоторого символа.
18. Дано предложение. Определить долю (в %) букв а в нем.
19. Дан текст. Сколько раз в нем встречается символ «+» и сколько раз символ «*»?
20. Дано предложение. Определить, сколько в нем одинаковых соседних букв.
21. Дано предложение. Определить:

- число вхождений в него буквосочетания ро;

- число вхождений в него некоторого буквосочетания из двух букв;
- число вхождений в него некоторого буквосочетания.

22. Дано предложение. В нем слова разделены одним пробелом (начальные и конечные пробелы и символ «-» в предложении отсутствуют). Определить количество слов в предложении.
23. Дано предложение. В нем слова разделены одним или несколькими пробелами (символ «-» в предложении отсутствует). Определить количество слов в предложении. Рассмотреть два случая:

- начальные и конечные пробелы в предложении отсутствуют;
- начальные и конечные пробелы в предложении имеются.

24. Дан текст. Подсчитать общее число вхождений в него символов «+» и «–».
25. Дан текст. Определить, сколько в нем предложений.
26. Дано предложение. Определить, сколько в нем гласных букв.
27. Дано предложение. Определить, какая из букв – о или а – встречается в нем чаще (принять, что указанные буквы в строке есть).
28. Дано предложение. Определить, есть ли в нем запятые.
29. Дано предложение. В нем слова разделены одним пробелом (символ «-» в предложении отсутствует). Верно ли, что число слов в предложении больше трех?
30. Дано предложение, в котором имеются буквы с и т. Определить, какая из них встречается позже (при просмотре слова слева направо). Если таких букв несколько, то должны учитываться последние из них. Оператор цикла с условием не использовать.
31. Дан текст. Верно ли, что в нем есть пять идущих подряд одинаковых символов?
32. Дано предложение. Напечатать все его символы, предшествующие первой запятой. Рассмотреть два случая:

- известно, что в предложении запятые имеются;
- в предложении запятых может не быть.

33. Дано предложение, в котором слова разделены двумя пробелами (начальных и конечных пробелов нет). Определить первое слово.
34. Дано предложение, в котором слова разделены одним пробелом (начальных и конечных пробелов нет). Определить последнее слово.
35. Дано предложение, в котором слова разделены двумя пробелами (начальных и конечных пробелов нет). Определить первое, второе и третье слова. Принять, что в заданном предложении есть не менее четырех слов.
36. Дано предложение, в котором слова разделены одним пробелом (начальных и конечных пробелов нет). Определить два последних слова. Принять, что в заданном предложении есть не менее трех слов.

Задание 2

Решить задачу согласно варианта, составить программу на языке программирования Python. Номер варианта необходимо получить у преподавателя.

1. Дано предложение. Определить, есть ли буква а в нем. В случае положительного ответа найти также порядковый номер первой из них.
2. Дано слово. Проверить, является ли оно палиндромом (палиндром читается одинаково в обоих направлениях, например «потоп»).
3. Дан текст. Определить количество букв и в первом предложении. Рассмотреть два случая:

- известно, что буквы и в этом предложении есть;
- букв и в тексте может не быть.

4. Дана последовательность символов, в начале которой имеется некоторое количество одинаковых символов. Определить это количество. Рассмотреть два случая:

- известно, что не все символы последовательности одинаковые;
- все символы последовательности могут быть одинаковыми.

5. Даны два слова. Определить, сколько начальных букв первого слова совпадает с начальными буквами второго слова. Рассмотреть два случая:

- известно, что слова разные;
- слова могут быть одинаковыми.

6. Дано предложение, в котором нет символа «-». Определить количество букв о в первом слове. Учесть, что в начале предложения могут быть пробелы.
7. Дано предложение. Определить количество букв н, предшествующих первой запятой предложения. Рассмотреть два случая:

- известно, что запятые в предложении есть;
- запятых в предложении может не быть.

8. Дано предложение. Определить порядковые номера первой пары одинаковых соседних символов. Если таких символов нет, то должно быть напечатано соответствующее сообщение.
9. Дано предложение. Определить, есть ли в нем буквосочетания чу или щу. В случае положительного ответа найти также порядковый номер первой буквы первого из них.
10. Дана последовательность слов. Проверить, правильно ли в ней записаны буквосочетания жи и ши.
11. Дана последовательность слов. Проверить, правильно ли в ней записаны буквосочетания ча и ща. Исправить ошибки.
12. Дано предложение. Напечатать все символы, расположенные между первой и второй запятыми. Если второй запятой нет, то должны быть напечатаны все символы, расположенные после единственной имеющейся запятой.
13. Дано предложение. Определить, какая из букв – н или к – встречается в ней раньше при просмотре слева направо (принять, что указанные буквы в строке есть).
14. Дана строка, в которой есть слово или. Определить, сколько раз оно встречается.
15. Дано предложение. Все его символы, стоящие на четных местах, заменить буквой ы.
16. Дано предложение. Все его символы, стоящие на третьем, шестом, девятом и т. д. местах, заменить буквой а.
17. Дано предложение. Заменить в нем все вхождения буквосочетания ах на ух.
18. Дано предложение. Заменить в нем все вхождения буквосочетания да на не.
19. Дано предложение. Заменить в нем все вхождения буквосочетания про на нет.
20. Символьной строке s по ошибке вместо опечатка присвоено значение очепатка. Изменить значение s так, чтобы ошибки не было.
21. Дано слово. Поменять местами его вторую и пятую буквы.
22. Дано слово. Поменять местами его третью и последнюю буквы.
23. Дано слово. Поменять местами его m-ю и n-ю буквы.
24. Дано слово из четного числа букв. Поменять местами первую букву со второй, третью – с четвертой и т. д.
25. Дано слово из четного числа букв. Поменять местами его половины следующим способом: первую букву поменять с последней, вторую – с предпоследней и т. д.
26. Дано слово из 12 букв. Переставить в обратном порядке буквы, расположенные между второй и десятой буквами (т. е. с третьей по девятую).
27. Дано слово из 15 букв. Переставить в обратном порядке буквы, расположенные между k-й и s-й буквами (т. е. с $(k+1)$-й по $(s-1)$-ю). Значения k и s вводятся с клавиатуры, $k < s$.

28. Дано слово. Поменять местами первую из букв а и последнюю из букв о. Учесть возможность того, что таких букв в слове может не быть.

Задание 3

Решить задачу согласно варианта, составить программу на языке программирования Python. Номер варианта необходимо получить у преподавателя.

1. Дано слово.

- Удалить из него третью букву.
- Удалить из него k-ю букву.

2. Дано слово.

- Удалить из него первую из букв о, если такая буква есть.
- Удалить из него последнюю из букв л, если такая буква есть.

4. Дано предложение. Удалить из него все символы с n_1-го по n_2-й ($n_1 \leq n_2$).
5. Дано предложение. Удалить из него все буквы с (как в кириллице так и на латинице).
6. Дана строка. Удалить из нее все пробелы.
7. Дано слово. Удалить из него все повторяющиеся буквы, оставив их первые вхождения, т. е. в слове должны остаться только различные буквы.
8. Дано предложение. Удалить из него все буквы о, стоящие на нечетных местах.
9. Дано слово, оканчивающее символом «.». Составить программу, которая вставляет некоторую заданную букву после буквы с заданным номером.
10. Дано слово, оканчивающее символом «.». Вставить заданную букву после первой буквы и.
11. Дано предложение, оканчивающее символом «.». Вставить заданную букву перед последней буквой и.
12. Путем вставок и удаления символов исправить ошибки:

- в слове прроцесор;
- во фразе теекстовыйфайл;
- во фразе програма и аллгоритм;
- во фразе процесор и паммять.

13. Дано ошибочно написанное слово рпроцессо. Путем перемещения его букв получить слово процессор.
14. Дано слово. Переставить его первую букву на место последней. При этом вторую, третью, ..., последнюю буквы сдвинуть влево на одну позицию.
15. Дано ошибочно написанное слово иинформаця. Путем перемещения его букв получить слово информация.
16. Дано слово. Переставить его первую букву на место k-й. При этом вторую, третью, ..., k-ю буквы сдвинуть влево на одну позицию.
17. Дано ошибочно написанное слово алигортм. Путем перемещения его букв получить слово алгоритм.
18. Дано слово. Переставить его s-ю букву на место k-й ($s < k$). При этом $(s + 1)$-ю, $(s + 2)$-ю, ..., k-ю буквы сдвинуть влево на одну позицию.
19. Дано ошибочно написанное слово роцессорп. Путем перемещения его букв получить слово процессор.
20. Дано слово. Переставить его последнюю букву на место первой. При этом первую, вторую, ..., предпоследнюю буквы сдвинуть вправо на одну позицию.
21. Дано ошибочно написанное слово ИТЕРНЕТН. Путем перемещения его букв получить слово ИНТЕРНЕТ.
22. Дано слово. Переставить его последнюю букву на место k-й. При этом k-ю, $(k + 1)$-ю, ..., предпоследнюю буквы сдвинуть вправо на одну позицию.

23. Дано ошибочно написанное слово килбайот. Путем перемещения его букв получить слово килобайт.
24. Дано слово. Переставить его s-ю букву на место k-й ($s > k$). При этом k-ю, $(k+1)$-ю, ..., $(s-1)$-ю буквы сдвинуть вправо на одну позицию.
25. Дано слово из 12 букв. Переставить его буквы следующим способом: первая, двенадцатая, вторая, одиннадцатая, ..., пятая, восьмая, шестая, седьмая.
26. Дана строка, состоящая только из букв. Заменить все буквы «а» на буквы «б» и наоборот, как заглавные, так и строчные. Например, при вводе строки «абвАБВ» должен получиться результат «бавБАВ».
27. Дан текст. Напечатать все имеющиеся в нем цифры.
28. Дан текст. Определить количество цифр в нем.
29. Дан текст, в котором имеются цифры.

- Найти их сумму.
- Найти максимальную цифру.

30. Дан текст, в начале которого имеются пробелы и в котором имеются цифры. Найти порядковый номер максимальной цифры, начиная счет с первого символа, не являющегося пробелом. Если максимальных цифр несколько, то должен быть найден номер первой из них.
31. Дан текст. Определить, является ли он правильной десятичной записью целого числа.
32. Дан текст, представляющий собой десятичную запись целого числа. Вычислить сумму цифр этого числа.
33. Дан текст, имеющий вид: «$d_1 + d_2 + \ldots + d_n$», где d_i – цифры ($n > 1$). Вычислить записанную в тексте сумму.
34. Дан текст, имеющий вид: «$d_1 - d_2 + d_3 - \ldots$», где d_i – цифры ($n > 1$). Вычислить записанную в тексте алгебраическую сумму.
35. Дан текст, имеющий вид: «$d_1 \pm d_2 \pm \ldots \pm d_n$», где di – цифры ($n > 1$). Вычислить записанную в тексте алгебраическую сумму.
36. Дан текст. Найти наибольшее количество идущих подряд цифр.
37. Дан текст, в котором имеется несколько идущих подряд цифр. Получить число, образованное этими цифрами.
38. Дан текст. Найти сумму всех имеющихся в нем чисел.
39. Дан текст. Найти максимальное из имеющихся в нем чисел.
40. Определить количество цифр в заданном натуральном числе, не выделяя каждую отдельную цифру.
41. Дано положительное вещественное число. Определить количество цифр:

- в его целой части;
- в его дробной части.
  Функции для работы с вещественными числами не использовать.

42. Дана строка, в которой без пробелов записано арифметическое выражение в виде суммы трех натуральных чисел, sнапример «1+25+3». Вычислить эту сумму.

Задание повышенной сложности

Составить программу для решения следующих задач. Номер варианта выбирается по согласованию с преподавателем.

1. Дано предложение. Найти наибольшее количество идущих подряд пробелов.
2. Дан текст. Найти наибольшее количество идущих подряд одинаковых символов.
3. Дано слово. Определить, сколько различных букв в нем.
4. В слове имеется только две одинаковые буквы. Найти их.

5. Даны два слова. Для каждой буквы первого слова (в том числе для повторяющихся в этом слове букв) определить, входит ли она во второе слово. Например, если заданные слова информация и процессор, то для букв первого из них ответом должно быть: нет нет нет да да нет нет да нет нет.
6. Даны два слова. Для каждой буквы первого слова определить, входит ли она во второе слово. Повторяющиеся буквы первого слова не рассматривать. Например, если заданные слова процессор и информация, то для букв первого из них ответом должно быть: нет да да да нет нет.
7. Даны два слова. Напечатать только те буквы слов, которые есть лишь в одном из них (в том числе повторяющиеся). Например, если заданные слова процессор и информация, то ответом должно быть: п е с с и ф м а я.
8. Даны два слова. Напечатать только те буквы слов, которые встречаются в обоих словах лишь один раз. Например, если заданные слова процессор и информация, то ответом должно быть: п е ф м а я.
9. Даны два слова. Определить, можно ли из букв первого из них получить второе. Рассмотреть два варианта:

- повторяющиеся буквы второго слова могут в первом слове не повторяться;
- каждая буква второго слова должна входить в первое слово столько же раз, сколько и во второе.

10. Даны три слова. Напечатать только те буквы слов, которые есть лишь в одном из слов. Рассмотреть два варианта:

- повторяющиеся буквы каждого слова рассматриваются;
- повторяющиеся буквы каждого слова не рассматриваются.

11. Даны три слова. Напечатать их общие буквы. Повторяющиеся буквы каждого слова не рассматривать.
12. Даны три слова. Напечатать неповторяющиеся в них буквы.
13. Дано предложение. Напечатать его в обратном порядке слов, например предложение мама мыла раму должно быть напечатано в виде раму мыла мама.
14. Дано предложение. Поменять местами его первое и последнее слова.
15. Дано предложение. Напечатать все его слова, отличные от слова привет.
16. Дано предложение. Определить количество слов:

- начинающихся с буквы н;
- оканчивающихся буквой р.

17. Дано предложение. Вывести на экран его слова:

- начинающиеся и оканчивающиеся на одну и ту же букву;
- которые содержат ровно три буквы е;
- которые содержат хотя бы одну букву о.

18. Дано предложение. Найти какое-нибудь его слово, начинающееся на букву к.
19. Дано предложение. Найти длину его самого короткого слова.
20. Дано предложение. Напечатать его самое длинное слово (принять, что такое слово – единственное).
21. Дано предложение. Верно ли, что его самое длинное слово имеет больше 10 символов?
22. Дано предложение. Напечатать все его слова в порядке неубывания их длин.
23. Дано предложение. Напечатать все слова, которые встречаются в нем по одному разу.
24. Дано предложение. Напечатать все его различные слова.
25. Дано предложение. В нем только два слова одинаковых. Найти эти слова.
26. Дано предложение. Напечатать все его слова, предварительно преобразовав каждое из них по следующему правилу:

- заменить первую встреченную букву а на о;
- удалить из слова все вхождения последней буквы (кроме нее самой);
- оставить в слове только первые вхождения каждой буквы;
- в самом длинном слове удалить среднюю (средние) букву(ы).
  Принять, что такое слово – единственное.

27. Дана последовательность слов. Напечатать те слова последовательности, которые отличны от первого слова и удовлетворяют следующему свойству:

- в слове нет повторяющихся букв;
- слово симметрично.

28. Даны два предложения. Для каждого слова первого предложения (в том числе для повторяющихся в этом предложении слов) определить, входит ли оно во второе предложение.
29. Даны два предложения. Для каждого слова первого предложения определить, входит ли оно во второе предложение. Повторяющиеся слова первого предложения не рассматривать.
30. Даны два предложения. Напечатать слова, которые есть только в одном из них (в том числе повторяющиеся).
31. Даны два предложения. Напечатать слова, которые встречаются в двух предложениях только один раз.
32. Дан текст. Проверить, правильно ли в нем расставлены круглые скобки (т. е. находится ли справа от каждой открывающей скобки соответствующая ей закрывающая скобка, а слева от каждой закрывающей – соответствующая ей закрывающая). Предполагается, что внутри каждой пары скобок нет других скобок.

- Ответом должны служить слова да или нет.
- В случае неправильности расстановки скобок: если имеются лишние правые (закрывающие) скобки, то выдать сообщение с указанием позиции первой такой
  скобки; если имеются лишние левые (открывающие) скобки, то выдать сообщение с указанием количества таких скобок. Если скобки расставлены правильно, то сообщить об этом.

33. Строка содержит арифметическое выражение, в котором используются круглые скобки, в том числе вложенные. Проверить, правильно ли в нем расставлены скобки.

- Ответом должны служить слова да или нет.
- В случае неправильности расстановки скобок: если имеются лишние правые (закрывающие) скобки, то выдать сообщение с указанием позиции первой такой
  скобки; если имеются лишние левые (открывающие) скобки, то выдать сообщение с указанием количества таких скобок. Если скобки расставлены правильно, то сообщить об этом.

34. Дано натуральное число n ($n \leq 1000$). Напечатать это число русскими словами (тринадцать, сто пять, двести сорок один, тысяча и т. д.).

Содержание отчета и его форма

Отчет по лабораторной работе оформляется электронно в формате PDF, должен содержать ответы на контрольные вопросы, ссылку на репозиторий с которым выполнялась работа, скриншоты IDE PyCharm, скриншоты результатов работы программ.

Вопросы для защиты работы

1. Что такое строки в языке Python?
2. Какие существуют способы задания строковых литералов в языке Python?

3. Какие операции и функции существуют для строк?
4. Как осуществляется индексирование строк?
5. Как осуществляется работа со срезами для строк?x
6. Почему строки Python относятся к неизменяемому типу данных?
7. Как проверить то, что каждое слово в строке начинается с заглавной буквы?
8. Как проверить строку на вхождение в неё другой строки?
9. Как найти индекс первого вхождения подстроки в строку?
10. Как подсчитать количество символов в строке?
11. Как подсчитать то, сколько раз определённый символ встречается в строке?
12. Что такое f-строки и как ими пользоваться?
13. Как найти подстроку в заданной части строки?
14. Как вставить содержимое переменной в строку, воспользовавшись методом format()?
15. Как узнать о том, что в строке содержатся только цифры?
16. Как разделить строку по заданному символу?
17. Как проверить строку на то, что она составлена только из строчных букв?
18. Как проверить то, что строка начинается со строчной буквы?
19. Можно ли в Python прибавить целое число к строке?
20. Как «перевернуть» строку?
21. Как объединить список строк в одну строку, элементы которой разделены дефисами?
22. Как привести всю строку к верхнему или нижнему регистру?
23. Как преобразовать первый и последний символы строки к верхнему регистру?
24. Как проверить строку на то, что она составлена только из прописных букв?
25. В какой ситуации вы воспользовались бы методом `splitlines()`?
26. Как в заданной строке заменить на что-либо все вхождения некоей подстроки?
27. Как проверить то, что строка начинается с заданной последовательности символов, или заканчивается заданной последовательностью символов?
28. Как узнать о том, что строка включает в себя только пробелы?
29. Что случится, если умножить некую строку на 3?
30. Как привести к верхнему регистру первый символ каждого слова в строке?
31. Как пользоваться методом `partition()`?
32. В каких ситуациях пользуются методом `rfind()`?