Глава 3 Строки и байтовые строки

§ 3.1 Строковый тип данных

3.1.1 Строки в Python

> **Строки** – это упорядоченные последовательности символов, для работы с текстовой информацией.

Строки в языке Python являются типом данных, специально предназначенным **для обработки текстовой информации**. Строка может содержать произвольно длинный текст (ограниченный имеющейся памятью).

Минимальное количество символов, из которого может состоять строка, – это ноль. Это легко проверить, выполнив такой код:

B [1]:

```
my_string = ''
len(my_string)
```

Out[1]:

```
0
```

Максимально допустимое количество элементов в строке зависит от технических характеристик компьютера, но в любом случае существующие ограничения скорее теоретические: в 32-разрядной версии Python предельно возможная длина строки составляет от 2 до 3 гигабайт (точная цифра зависит от конфигурации ОС). В 64-разрядной версии Python, если позволяет объем оперативной памяти, можно обрабатывать строки, содержащие 60 и более гигабайт.

Можно создать строку, состоящую из 747323 символов (примерно столько символов в произведении "Война и мир"). И она будет создана.

В новых версиях Python имеются два типа строк: ***обычные строки (последовательность байтов)*** и ***Unicode-строки (последовательность символов)***. В Unicode-строке каждый символ может занимать в памяти 2 или 4 байта, в зависимости от настроек периода компиляции. Четырехбайтовые знаки используются в основном для восточных языков.

> **Строка в Python** – это **неизменяемая** последовательность Unicod'ных символов. Для строк определен тип `str` .

По умолчанию Python хранит строки в кодировке UTF-8.

> ***UTF-8*** (Unicode Transformation Format, 8-bit – «формат преобразования Юникода, 8-бит») – распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Unicode, используя переменное количество байт (от 1 до 4), и обеспечивающий полную обратную совместимость с 7-битной кодировкой ASCII.

Чтобы определить строку её при записи необходимо заключить в **двойные кавычки** или **апострофы**. При этом важно, чтобы с обоих концов литерала использовались **кавычки одного и того же типа**. Также можно использовать строки в тройных кавычках, то есть строки, которые начинаются и заканчиваются тремя символами кавычки (либо тремя кавычками, либо тремя апострофами). С помощью тройных кавычек можно записывать строки, состоящие из нескольких абзацев.

In [1]:

```
# Строки могут быть записаны различными способами:

s1 = 'строка в одинарных кавычках'
s2 = "строка в двойных кавычках"
s3 = '''
многострочный
текст
'''
s4 = """
Ансамбли народной песни и пляски доказывают,
что наш народ необычайно богат людьми,
не умеющими ни петь, ни плясать.

Максим Звонарев
"""
```

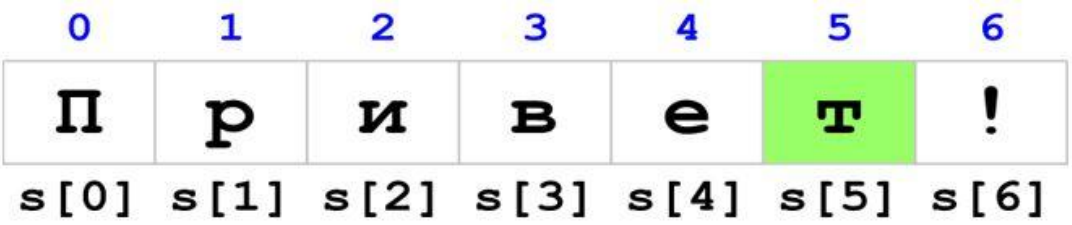
In [2]:

```
print(s4)
```

```
Ансамбли народной песни и пляски доказывают,
что наш народ необычайно богат людьми,
не умеющими ни петь, ни плясать.

Максим Звонарев
```

В строке символы **проиндексированы с нуля**. И для просмотра символа необходимо указать его индекс в квадратных скобках.



In [1]:

```
st = 'Привет!'
print(st[0])
print(st[-2])
```

```
П
т
```

Поскольку строки – неизменяемый тип данных, то операция `st[4] = 'a'` невозможна. Но можно составить новую строку из уже имеющихся: `st_new = st + '!'` .

In [33]:

```
st = 'python'
# st[0] = 'P' приведет к ошибке
'P' + st[1:]  # новая строка из уже имеющейся
```

Out[33]:

```
'Python'
```

Пример, демонстрирующий, что **строка – неизменяемый объект**.

In [25]:

```
st = 'Привет'
print(id(st))

st += ', Мир!'
print(id(st))
```

```
1988663203408
1988663357600
```

В примере создаем строку "*Привет*". Дальше печатаем её адрес в памяти. Адрес памяти у объекта в Python можно запросить с помощью встроенной функции `id` . Далее к строке добавляем ещё одну строчку "*, Мир!*", чтобы получилась строка "*Привет, Мир!*" и печатаем адрес получившегося строкового объекта.

Как видно, эти адреса отличаются. То есть когда мы изменили нашу начальную строку, мы на самом деле создали новый строковый объект.

В Питоне **отсутствует отдельный тип символа (*char*)**, вместо него используются строки единичной длины.

Для получения символа по его коду и кода по символу имеются соответствующие функции `chr()` и `ord()` :

In [6]:

```
print(chr(65), ord('A'))
print(chr(48), ord('0'))
```

```
A 65
0 48
```

3.1.2 Экранирование символов

Если в строке необходимо использовать специальные символы (например, перенос или одноименные кавычки), можно воспользоваться **механизмом экранирования символов**, для чего используется специальный символ .

Таблица. Некоторые из экранированных последовательностей

Последовательность	Значение
\\	Обратный слеш (\)
\'	Апостроф (')
\"	Кавычка (")
\n	Символ «Перевод строки»
\t	Символ «Горизонтальная табуляция»
\r	Символ «Возврат каретки»
\a	Звонок
\b	Символ «Забой»
\f	Символ «Перевод страницы»
\v	Символ «Вертикальная табуляция»

In [32]:

```
# Примеры экранирования символов
example_string = "Текст, который содержит \"текст в кавычках\""
print(example_string)

example_string = 'Текст, который содержит \'текст в апострофах\''
print(example_string)

example_string = "Если внутри кавычки другого типа, можно не 'экранировать'"
print(example_string)

example_string = "Так в строке будет табуляция\t\t\t и \nперенос, а также обратный слеш \
print(example_string)
```

```
Текст, который содержит "текст в кавычках"
Текст, который содержит 'текст в апострофах'
Если внутри кавычки другого типа, можно не 'экранировать'
Так в строке будет табуляция                    и
перенос, а также обратный слеш \
```

С помощью обратного слэша можно **разбить длинную строку** внутри вашего кода на несколько строк: записываете строку, ставите обратный слэш и переносите строчку на следующую строку кода (как видно

на примере ниже).

In [16]:

```
st = 'Крылатые слова, крылатые выражения — устойчивые, афористические, \
как правило, образные выражения, вошедшие в речь из литературно-художественных, \
публицистических, философских, фольклорных и других источников. \
К крылатым выражениям также относят популярные изречения исторических личностей. \
(Википедия)'
print(st)
```

```
Крылатые слова, крылатые выражения — устойчивые, афористические, как правил
о, образные выражения, вошедшие в речь из литературно-художественных, публиц
истических, философских, фольклорных и других источников. К крылатым выражен
иям также относят популярные изречения исторических личностей. (Википедия)
```

Резюме:

- можно двойные (одинарные) кавычки экранировать символом обратного слэша и, таким образом, двойные (одинарные) кавычки записывать внутри строки, которая создается с помощью двойных (одинарных) кавычек;
- использовать двойные кавычки внутри одинарных (и наоборот);
- можно выводить спец. символы (табуляция, обратный слеш и пр.).

3.1.3 Сырые *r*-строки

В Python'е есть **концепция сырых строк** – это строки, которые начинаются с английской буквы `r`. С помощью сырых строк можно объявить строку и не экранировать специальные символы, которые находятся внутри этой строки.

Пример. Вывод пути к директории Windows.

В коде ниже в первом случаи мы экранируем обратные слэши обратными слэшами и это выглядит достаточно некрасиво. Во втором случае мы используем сырые строки и просто записываем наш путь на диске.

In [22]:

```
st = "Путь к файлу на диске: G:\\Haskell\\Moduls"
print(st)

st = r"Путь к файлу на диске: G:\Haskell\Moduls"
print(st)
```

```
Путь к файлу на диске: G:\Haskell\Moduls
Путь к файлу на диске: G:\Haskell\Moduls
```

§ 3.2 Индексы. Срезы строк

Как и у любого **итерируемого объекта**, у каждого элемента строковой переменной есть свой индекс, по которому мы можем к нему обратиться. Индексация символов строки подчиняется тем же **правилам**, что и индексация элементов списка.

1. **Первый** символ строки имеет **индекс 0**.
2. Возможно использование **отрицательных индексов**. При этом последний символ в строке будет иметь индекс (-1), предпоследний (-2) и т.д.
3. Из строки можно выбрать несколько идущих друг за другом символов с помощью среза. **Для создания среза** необходимо указать имя переменной, индекс первого элемента среза и индекс элемента, стоящего после последнего элемента среза. Индексы должны быть заключены в квадратные скобки и разделены двоеточием. *Пример*: `st[2:5]` .
4. **Первый и последний элементы среза можно не указывать**, но двоеточие при этом следует оставить на своём месте! В этом случае в индекс будут включены все символы с начала строки, если не указан первый индекс, или все символы до конца строки, если не указан последний индекс. *Примеры*: `st[:]` – все символы строки; `st[3:]` – символы строки, начиная с четвертого и до конца; `st[:6]` – символы строки, от начала и включая шестой.
5. **Допускается использование индекса с шагом**. Шаг указывает интервал, с которым нужно выбирать символы из строки. *Пример*: `st[1::2]` – данный код выбирает из строки каждый второй символ.
6. **Шаг может быть отрицательным**. В этом случае элементы будут перебираться справа налево. Для корректной работы кода, в котором используется срез с отрицательным шагом, необходимо, чтобы первый индекс (индекс первого элемента среза) был больше второго (индекс элемента, следующего за последним). *Пример*: `st[10:4:-1]` .

Срезы строк используют для того, чтобы из целой строки выделить какую-то подстроку.

Синтаксис срезов: `строка[start:stop:step]`

> **Срез** – это квадратные скобки и внутри три параметра: `start, stop, step` . Здесь `start` – номер символа в строке, начиная с которого мы хотим получить подстроку (нумерация начинается с нуля), `stop` – номер символа до которого выделяется подстрока (без него самого), `step` – шаг с которым выделяются символы из строки.

In [31]:

```
st = '0123456789'
print(st[::-1])       # "9876543210" - реверс строки
print(st[:-2])        # "01234567"
print(st[-6:-2])      # "4567"
print(st[::2])        # "02468"
```

```
9876543210
01234567
4567
02468
```

§ 3.3 Операции над строками

Для объединения двух строк используется оператор сложения (знак `+`). В итоге получается новая строка. Когда вы складываете много строчек, это может быть не эффективно, потому что строки неизменяемые и при сложении нескольких строк на каждое сложение создается новый строковый объект и создается новая локация памяти. Однако, в самом простейшем случае вы можете просто сложить две строки и это будет решением множества практических задач.

Строки можно даже умножать, потому что для них переопределен оператор умножения (знак `*`). Если вы умножите строчку на три, например, то получите ту же самую строчку, повторяющуюся три раза.

In [36]:

```
st = 'Можно просто ' + 'складывать строки'
print(st)

st = 'Их можно даже умножать! ' * 10
print(st)
```

```
Можно просто складывать строки
Их можно даже умножать! Их можно даже умножать! Их можно даже умножать! Их м
ожно даже умножать! Их можно даже умножать! Их можно даже умножать! Их можно
даже умножать! Их можно даже умножать! Их можно даже умножать! Их можно даже
умножать!
```

Так как в Python всё есть объект, **у строк есть методы**. Если мы определим строку и присвоим её переменой, то у этой строки мы потом можем вызывать различные методы. Строки поддерживают все общие операции для последовательностей, и имеют ряд дополнительных методов (все методы можно посмотреть в документации).

Таблица. Методы строк

Метод	Описание метода
`x in s`	Проверка на наличие `x` в `s`
`x not in s`	Проверка на отсутствие `x` в `s`
`max(s)`, `min(s)`	Максимальный/минимальный элемент в `s`
`chr(i)`	Возвращает символ с кодом i из таблицы Unicode
`ord(c)`	Возвращает код символа `c` из таблицы Unicode
`len(s)`	Возвращает длину строки `s`
`s.upper()`	Возвращает копию строки `s` в верхнем регистре
`s.lower()`	Возвращает копию строки `s` в нижнем регистре
`s.capitalize()`	Возвращает копию строки `s` с первым символом в верхнем регистре (делает первую букву в строке заглавной)
`s.swapcase()`	Возвращает копию строки `s`, в которой символы верхнего регистра преобразованы в нижний регистр, а символы нижнего – в верхний
`s.title()`	Возвращает копию строки `s`, в которой первые символы каждого слова преобразованы в верхний регистр, а все остальные – в нижний регистр
`s.count(t[, start[, end]])`	Возвращает число вхождений строки `t` в строку `s` (или в срез `s[start:end]`)

Метод	Описание метода
`s.find(t[, start[, end]])`	Поиск подстроки в строке: возвращает позицию самого первого (крайнего слева) вхождения подстроки `t` в строку `s` (или в срез `s[start:end]`); если подстрока `t` не найдена, возвращается `-1`
`s.rfind(t[, start[, end]])`	Поиск подстроки в строке `s`: возвращает номер последнего вхождения или `-1`
`s.index(t[, start[, end]])`	Аналогично `s.find()`, но генерируется исключение `ValueError`, если подстрока не найдена
`s.rindex(t[, start[, end]])`	Аналогично `s.rfind()`, но генерируется исключение `ValueError`, если подстрока не найдена
`s.replace(old, new[, count])`	Возвращает копию строки `s`, в которой каждое (но не более `count`, если этот аргумент определен) вхождение подстроки `old` замещается подстрокой `new`
`s.split(sep=None, maxsplit=-1)`	Возвращает список строк, полученный при разбиении строки `s` по строке `sep`
`s.join(seq)`	Возвращает строку-«склейку» элементов `seq`, используя `s` в качестве разделителя
`s.isdigit()`	Позволяет проверить является ли строка `s`, у которой мы вызвали этот метод, числом (строка состоит из цифр) и возвращает `boolean` (`True` или `False`)
`s.isalpha()`	Проверяет состоит ли строка `s`, у которой мы вызвали этот метод, только из букв и возвращает `boolean` (`True` или `False`)
`s.isalnum()`	Проверяет состоит ли строка `s`, у которой мы вызвали этот метод, из букв и цифр и возвращает `boolean` (`True` или `False`)
`s.islower()`	Проверка того, что все символы строки `s` в нижнем регистре и возврат `boolean` (`True` или `False`)
`s.isupper()`	Проверка того, что все символы строки `s`, у которой мы вызвали этот метод, находятся в верхнем регистре. Возвращает `boolean` (`True` или `False`)
`s.istitle()`	Проверка того, что строка `s` начинается с прописного символа. Возвращает `boolean` (`True` или `False`)
`s.isspace()`	Проверка того, что строка `s` содержит специальные символы: пробел, символ перевода страницы (`\f`), "новая строка" (`\n`), "перевод каретки" (`\r`), "горизонтальная табуляция" (`\t`) и "вертикальная табуляция" (`\v`). Возвращает `boolean` (`True` или `False`)
`s.startswith(t)`	Проверка того, что строка `s` начинается со строки `t`. Возвращает `boolean` (`True` или `False`)
`s.endswith(t)`	Проверка того, что строка `s` заканчивается шаблоном. Возвращает `boolean` (`True` или `False`)
`s.center(длина [, символ])`	Выводит по центру строку `s` заданной длины, где по краям стоят символы (по умолчанию пробел)
`s.lstrip([символ])`	Удаляет символы в начале строки `s` (по умолчанию удаляемый символ — пробел)
`s.rstrip([символ])`	Удаляет символы в конце строки `s` (по умолчанию удаляемый символ — пробел)
`s.strip([символ])`	Удаляет символы с начала и конца строки `s` (по умолчанию удаляемый символ — пробел)
`s.ljust(width, fillchar='символ')`	Создает строку `s` длины не меньшей, чем `width`, заполняя остальное заданным символом
`s.rjust(width, fillchar='символ')`	Создает строку `s` длины не меньшей, чем `width`, заполняя первые позиции заданным символом
`s.splitlines(num=string.count('\n'))`	Разбивает строку `s` на все (или числа) строки и возвращает список каждой строки с удаленными символами новой строки

In [37]:

```
s = "ЭТО просТо ТеКст"
ord(s[0])       # 1069
```

Out[37]:

```
1069
```

In [38]:

```
chr(1069)       # 'Э'
```

Out[38]:

```
'Э'
```

In [44]:

```
s.upper(), s.lower(), s.title(), s.capitalize(), s.swapcase()
            # ('ЭТО ПРОСТО ТЕКСТ', 'это просто текст', 'Это Просто Текст', 'Это просто те
```

Out[44]:

```
('ЭТО ПРОСТО ТЕКСТ',
 'это просто текст',
 'Это Просто Текст',
 'Это просто текст',
 'это ПРОСтО тЕкСТ')
```

In [46]:

```
"Это просто текст".count("о")      # 3
```

Out[46]:

```
3
```

In [47]:

```
"ЭТО просТо ТеКст".find("Т")       # 1
```

Out[47]:

```
1
```

In [41]:

```
"ЭТО просТо ТеКст".replace("Т", "т")      # 'ЭтО просто теКст'
```

Out[41]:

```
'ЭтО просто теКст'
```

In [52]:

```
"МОРОЖЕНОЕ".replace("О", "У", 2)
```

Out[52]:

```
'МУРУЖЕНОЕ'
```

In [42]:

```
"ЭТО просТо ТеКст".split()                    # ['ЭТО', 'просТо', 'ТеКст']
```

Out[42]:

```
['ЭТО', 'просТо', 'ТеКст']
```

In [49]:

```
lst = ['Мир', 'Труд', 'Май']
"! ".join(lst) + '!'          # Мир! Труд! Май!
```

Out[49]:

```
'Мир! Труд! Май!'
```

In [6]:

```
'124'.isdigit(), 'asD'.isalpha(), 'aqsx1HG223'.isalnum()
```

Out[6]:

```
(True, True, True)
```

In [8]:

```
'Это вопросительное предложение?'.endswith('?')
```

Out[8]:

```
True
```

In [1]:

```
print("Анкета слушателя".center(50, "*"))
# **********Анкета слушателя**********

print("Анкета слушателя". ljust (50, "-"))
# Анкета слушателя----------

print("Анкета слушателя". rjust (50, "~"))
# ----------Анкета слушателя
```

```
**********Анкета слушателя**********
Анкета слушателя----------
~~~~~~~~~~Анкета слушателя
```

In [64]:

```
# Пример использования split()
st = 'Кащеев Бармалей Лешиевич'      # Простая строка
fio = st.split()                     # Извлечение по пробелам
print(fio)

name = fio[1]
print(name)
```

```
['Кащеев', 'Бармалей', 'Лешиевич']
Бармалей
```

In [65]:

```
st = '!!!!!!Всюду ! - плохо!!!!!!!!'
st.strip('!')
```

Out[65]:

```
'Всюду ! - плохо'
```

In [70]:

```
st = "это первая строка \na это вторая строка \nтак можно дальша продолжать,\nно здесь по
print (st.splitlines( ))
```

```
['это первая строка ', 'а это вторая строка ', 'так можно дальша продолжат
ь,', 'но здесь последняя строка']
```

§ 3.4 Оператор `in`

Оператор `in` позволяет проверить наличие подстроки в строке. Например, мы можем записать вот такую вещь и проверить, что 3.14 содержится в строке 'Число pi = 3.1415926'. В данном случае она содержится, поэтому результат `True`, то есть истина. Если мы проверим наличие подстроки, которая не содержится в строке, мы получаем `False`.

In [72]:

```
'3.14' in 'Число pi = 3.1415926'
```

Out[72]:

```
True
```

§ 3.5 Оператор `for...in`

Выражение `for...in` позволяет итерироватся по строке. Известно, что строка – это последовательность юникодных символов. Соответственно итерация по строке означает взятие каждого следующего элемента по очереди.

Пример. Если мы объявим строковый объект изначальный "привет" и дальше проитерируемся по этой строке, используя выражение `for...in`, то мы можем получить каждый отдельный символ этой строки – букву П, букву Р, букву И и т.д.

In [73]:

```
st = 'Привет'
for letter in st:
    print('Буква', letter)
```

```
Буква П
Буква р
Буква и
Буква в
Буква е
Буква т
```

§ 3.6 Конвертация типов

Конвертацию типов можно использовать при преобразовании строки в число и числа в строку. Например, у нас есть вещественное число, и мы можем в Runtime (в процесе работы Python) применить конвертацию типов и преобразовать вещественное число в число типа `str`.

In [7]:

```
# Пример преобразования числа в строку
num = 999.01
print(type(num))

st = str(num)
print(st, type(st))
```

```
<class 'float'>
999.01 <class 'str'>
```

Мы берем 999.01, преобразовываем в строку и проверяем, что у нас действительно получилась в результате конвертации типов строка.

In [9]:

```
# Пример преобразования строки в число:
st = "123.456"
num = float(st)
print(num, type(num))

st = "123"
num = int(st)
print(num, type(num))
```

```
123.456 <class 'float'>
123 <class 'int'>
```

Если преобразовать не пустую строку к логическому типу, то мы получим `True`. Пустая строка, которая не содержит ни одного символа в себе, при конвертации к логическому типу возвращает `False`.

In [12]:

```
st = 'Не пустая строка'
num = bool(st)
print(num, type(num))

st = ''
num = bool(st)
print(num, type(num))
```

```
True <class 'bool'>
False <class 'bool'>
```

§ 3.7 Форматирование строк в Python

> **Форматирование строк** – это процесс, когда у вас есть какой-то строковый шаблон, внутрь которого вместо определенных ***placeholder-ов*** (заполнителей) вы хотите подставить те или иные значения из ваших переменных.

3.7.1 Стандартный подход

Стандартный подход предполагает **использование стандартного оператора** `%` . Операция форматирования заменяет в строке форматирования все записи, начинающиеся с `%` и оканчивающиеся буквой, на данные, указанные справа от операции, по очереди. Эта операция позволяет выводить текст с включениями чисел, строк и других данных в поля определенной ширины, выровненные по левому или правому краю. Для чисел строка форматирования дает возможность указывать количество цифр после запятой. Другими словами, оператор используется, когда в строку нужно добавить некоторые элементы строки, которые принимают определенные значения в процессе выполнения программы.

Структура оператора имеет вид

```
%<набор спецификаторов> % <что нужно форматировать>
```

Спецификатор `s` сообщает обработчику, что в качестве подставляемого значения будет передана строка. В качестве спецификатора используются:

- `%d` , `%i` , `%u` — десятичное число;
- `%f` — число с плавающей точкой;
- `%%` — будет выведен процент;
- `%o` — число в восьмеричной системе;
- `%x` , `%X` — 16-ричная система в нижнем и верхнем регистре;
- `%e` , `%E` — экспоненциальное представление числа с плавающей точкой;
- `%c` — код символа.

В [15]:

```
fio = "Иванов И.О."
sr_ball = 4.78
number = 178
print("Абитуриент %s (рег.номер - %d) имеет средний балл, равный %.2f руб." % (fio, numbe
```

```
Абитуриент Иванов И.О. (рег.номер - 178) имеет средний балл, равный 4.78 ру
б.
```

В [13]:

```
x = 2365
print("Перевод %d в разные СС: %o (8СС), %X (16СС)" % (x, x, x))
```

```
Перевод 2365 в разные СС: 4475 (8СС), 93D (16СС)
```

3.7.2 Использование метода `format()`

Второй способ форматирования строк – это **использование метода** `format()` , который есть у строки и который позволяет удобно формировать строку из комбинации значений различного типа.

Обычно под форматом чисел понимают форму их представления для пользователя, например число знаков до и после запятой, центрирование на экране и т.п. Поэтому инструкция `format()` используется в качестве метода инструкции `print()` .

Синтаксис инструкции `format` можно представить в следующем виде:

```
print("{: опции}".format(число))
```

Часто используемые **опции**, которые не являются обязательными, приведены ниже:

- символ заполнителя (например `*`);
- символ выравнивания `<` по левому краю, `>` по правому краю, `^` по центру, `=` требование заполнять пространство между знаком числа и его первой значащей цифрой;
- "+" – обязательный вывод знака числа;
- "-" – вывод знака только для отрицательных чисел;
- " " – пробел для положительного числа и "-" – для отрицательного числа.

Далее указывается число цифр, которые будут выводиться на экран.

Для вещественных чисел можно указать общее число цифр и число знаков после запятой в формате `X.Xf`. Если требуется вывести число в виде мантиссы, то формат будет `X.Xg`. Здесь после точки указывается число значащих цифр. Если использовать символ `e`, то данные будут выводиться в экспоненциальном формате `X.Xe`. В этом случае после точки ставится количество знаков в дробной части числа.

Можно указать **тип системы счисления**:

- `b` – двоичная;
- `o` – восьмеричная;
- `x` – 16-ричная в нижнем регистра;
- `X` – 16-ричная в верхнем регистре;
- `d` – десятичная системы счисления.

Если перед символом типа системы счисления указать символ `#`, то будет выводиться тип системы счисления.

Можно применять форматирование для нескольких чисел:

```
print ("{:опции1}] [{:опции2}".format(число1, число2))
```

In [17]:

```
a = 125
b = 156.00236
c = 178
print("При a={0:6.2f} b={1:<10.2e}. В 16СС: c={2:X}, a={0:x}".format(a, b, c))
print("Перевод {0: d} в разные СС: {0: #b}, {0: #o}, {0: #X}".format(198))
```

```
При a=125.00 b=1.56e+02  . В 16СС: c=B2, a=7d
Перевод  198 в разные СС:  0b11000110,  0o306,  0XC6
```

Другой способ, чуть более наглядный, опять же с **использованием функции `format()`, но с именованными аргументами**. В данном случае, у нас внутри *placeholder*-а содержится имя этих *placeholder*-ов, а в функцию `format` мы передаем, соответственно, именованные аргументы по имени, совпадающему с именем *placeholder*-а и получаем итоговую строку.

In [25]:

```
print('Давайте знакомиться. Я - {name}, мне {age} лет.'.format(age=18, name='Маша'))

name = 'Маша'
age = 18
print('Давайте знакомиться. Я - {}, мне {} лет.'.format(name, age))
print('Давайте знакомиться. Я - {0}, мне {1} лет.'.format(name, age))
```

```
Давайте знакомиться. Я - Маша, мне 18 лет.
Давайте знакомиться. Я - Маша, мне 18 лет.
Давайте знакомиться. Я - Маша, мне 18 лет.
```

In [28]:

```
# Примеры форматирования с использованием функции format
N = 123
print("|{:5d}".format(N))

X = 123.456
print("|{:7.2f}".format(X))
print("|{:10.2e}".format(X))
```

```
|   123
| 123.46
|  1.23e+02
```

3.7.3 `f`-строки

Третий способ форматирования (появился в Питоне 3.6) – это так называемые **`f`-строки**. Это строки, которые начинаются с префикса английской буквы `f`. Посмотрите на пример – мы можем просто объявить две переменные, а затем записать `f`-строку, в которой внутри *placeholder*-ов просто написать имена этих переменных. И вместо этих *placeholder*-ов будут подставлены правильные значения.

In [29]:

```
name = 'Маша'
age = 18

print(f'Давайте знакомиться. Я - {name}, мне {age} лет.')
```

```
Давайте знакомиться. Я - Маша, мне 18 лет.
```

В `f`-строках можно использовать **модификаторы форматирования**, позволяющие сообщить языку, как вы хотите то или иное значение в *placeholder*-е подставлять, в каком виде его в итоге вывести. Модификаторов форматирования очень много, почитать про них подробно можно в документации.

In [32]:

```
x = 56/3
print(f'x = {x:.3f}')

num = 127
print(f'num = {num:#b}')
```

```
x = 18.667
num = 0b1111111
```

§ 3.8 Рекомендации по эффективности. Примеры программ

При работе с очень длинными строками или большим количеством строк, **применяемые операции могут по-разному влиять на быстродействие программы**. Например, не рекомендуется многократно использовать операцию конкатенации для склеивания большого количества строк в одну. Лучше накапливать строки в списке, а затем с помощью `join()` собирать в одну строку.

In [36]:

```
a = ""
s = "Это строка"
for c in s:
    a += c  # неэффективно!
print(a)
```

```
Это строка
```

In [34]:

```
a = ""
s = "Это строка"
a = "".join([c for c in s]) # более эффективно
print(a)
```

```
Это строка
```

Пример. Найти сумму цифр в веденной строке.

In [19]:

```
# Найти сумму цифр в веденной строке

st = input()
summa = sum([int(c) for c in st if c.isdigit()])
print(summa)
```

```
65f4v
15
```

Пример. Проверить,является ли введенная строка идентификатором.

В [18]:

```
# Является ли строка идентификатором

s = input()
print(f'{s} is a valid identifier = {s.isidentifier()}')
```

```
_laba1
_laba1 is a valid identifier = True
```

Пример. Вводится строка, включающая строчные и прописные буквы. Требуется вывести ту же строку, заменив в ней строчные буквы прописными, а прописные – строчными. Например, исходная строка – "aB!cDEf", новая строка – "Ab!CdeF". В коде используйте цикл `for`, строковые методы `upper()` (преобразование к верхнему регистру) и `lower()` (преобразование к нижнему регистру), а также методы `isupper()` и `islower()`, проверяющие регистр строки или символа.

В [42]:

```
# Решение 1
# Создаем новую строку, т.к. строки - не изменяемые объекты

st = input("Введите строку: ")
new_st = ""

for c in st:
    if c.isupper():
        new_st += c.lower()
    else:
        new_st += c.upper()

print("Исходная строка:", st, sep="\n")
print("Новая строка:", new_st, sep="\n")
```

```
Введите строку: sdcLJlkjLKJL
Исходная строка:
sdcLJlkjLKJL
Новая строка:
SDCljLKJlkjl
```

В [45]:

```
# Решение 2
# Используем списки, т.к. списки менять можно

st = list(input("Введите строку: "))

for i in range(len(st)):
    if st[i].isupper():
        st[i] = st[i].lower()
    else:
        st[i] = st[i].upper()

new_st = ''.join(st)
print("Новая строка:", new_st)
```

```
Введите строку: ersdKJkjKJ
Новая строка: ERSDkjKJkj
```

В [46]:

```
# Решение 2
# Используем метод строки

st = input('Введите строку ')
st = st.swapcase()
print("Новая строка:", st)
```

```
Введите строку erJHLJLljl
Новая строка: ERjhljlLJL
```

Пример (ЕГЭ по информатике). Исполнитель Редактор получает на вход строку цифр и преобразовывает её. Редактор может вы-полнять две команды, в обеих командах `v` и `w` обозначают цепочки символов.

1. `заменить (v, w)`
2. `нашлось (v)`

Первая команда заменяет в строке первое слева вхождение цепочки `v` на цепочку `w`. Если цепочки `v` в строке нет, эта команда не изменяет строку. Вторая команда проверяет, встречается ли цепочка `v` в строке исполнителя Редактор. Если она встречается, то команда возвращает логическое значение "истина", в противном случае возвращает значение "ложь". Дана программа для исполнителя Редактор:

```
НАЧАЛО
ПОКА нашлось (222)
    заменить (222, 1)
    заменить (111, 2)
КОНЕЦ ПОКА
КОНЕЦ
```

Какая строка получится в результате применения приведённой программы к строке вида $1\ldots12\ldots2$ (2019 единиц и 2019 двоек)?

Ответ: 21

В [51]:

```
st = '1' * 2019 + '2' * 2019
while '222' in st:
    st = st.replace('222', '1', 1)
    st = st.replace('111', '2', 1)
print(st)
```

```
21
```

Пример. Напишите код, который обрабатывает строковые данные и возвращает их с первыми заглавными буквами в каждом слове. Примечание: Слова в строке отделены пробелами. Каждое слово после обработки должно быть с заглавной буквы, только, если первый символ слова - буква, в случае типа `3amg` код не должен менять буквы.

Подставьте "Входные данные" в свою программу и сравните результат с выходными данными.

Номер теста	Входная строка	Результат работы программы
Тест 1	`ivan ivanov`	`Ivan Ivanov`
Тест 2	`can you solve it?`	`Can You Solve It?`
Тест 3	`abracadabra`	`Abracadabra`
Тест 4	`a1 2b 3 abc d3e r2D2`	`A1 2b 3 Abc D3e R2D2`

В [47]:

```
st = input()

newst = st[0].upper()
for i in range(1, len(st)):
    if st[i-1] == ' ' and st[i].islower:
        newst += st[i].upper()
    else: newst += st[i]

print(newst)
```

```
a1 2b 3 abc d3e r2D2
A1 2b 3 Abc D3e R2D2
```

§ 3.9 Контрольное задание. Обработка строк

Задание. Пароль считается надежным, если его длина составляет не менее 12 символов, при этом он должен содержать хотя бы одну заглавную букву, хотя бы одну строчную букву, хотя бы одну цифру и хотя бы один спецсимвол. Напишите код, который обрабатывает строковые данные и выводит сообщение о том, надежен ли пароль или нет. В случае, если пароль не надежен, код должен выдавать рекомендации для усиления надежности пароля.

Замечание.

- Цифры, которые можно использовать для пароля: `1234567890` .
- Строчные буквы, которые можно использовать для пароля: `abcdefghijklmnopqrstuvwxyz`
- Заглавные буквы, которые можно использовать для пароля: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`
- Спецсимволы, которые можно использовать для пароля: `'!@#$%^&*()-+'`

- На данном этапе еще не введено понятие "Функция", поэтому необходимо создать файл и написать код, как в предыдущих разделах. Сначала необходимо объявить переменные, затем написать код, удовлетворяющий условию задания.

Подставьте "Входные данные" в свою программу и сравните результат с выходными данными.

Номер теста	Входная строка	Результат работы программы
Тест 1	`qwerty`	`Слабый пароль. Рекомендации: увеличить число символов - 6, добавить 1 заглавную букву, добавить 1 цифру, добавить 1 спецсимвол`
Тест 2	`Qwert_Y`	`Ошибка. Запрещенный спецсимвол`
Тест 3	`Q123wer123tY`	`Слабый пароль. Рекомендации: добавить 1 спецсимвол`
Тест 4	`@PowerRangers123@`	`Пароль надежен.`

В [7]:

```
password = input()
AZ = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
az = 'abcdefghijklmnopqrstuvwxyz'
dig = '0123456789'
spesial = '!@#$%^&*()-+'

if len(password) >= 12:
    valid_len = True
else:
    valid_len = False

valid_AZ = False
valid_az = False
valid_dig = False
valid_spesial = False
error = False

for c in password:
    if c in AZ:
        valid_AZ = True
    if c in az:
        valid_az = True
    if c in dig:
        valid_dig = True
    if c in spesial:
        valid_spesial = True
    if c not in(AZ + az + dig + spesial):
        error = True

if valid_len and valid_AZ and valid_az and valid_dig and valid_spesial and not error:
    print('Пароль надежен.')
else:
    if error:
        print('Ошибка. Запрещенный спецсимвол.')
    else:
        print('Пароль ненадежен. Рекомендации: ', end='')
        if not(valid_len):
            print('увеличить число символов -', len(password), end='')
        if not(valid_AZ):
            if not(valid_len):
                print(', ', end='')
            print('добавить 1 заглавную букву', end='')
        if not(valid_az):
            if not(valid_len) or not(valid_AZ):
                print(', ', end='')
            print('добавить 1 строчную букву', end='')
        if not(valid_dig):
            if not(valid_len) or not(valid_AZ) or not(valid_az):
                print(', ', end='')
            print('добавить 1 цифру', end='')
        if not(valid_spesial):
            if not(valid_len) or not(valid_AZ) or not(valid_az) or not(valid_dig):
                print(', ', end='')
            print('добавить 1 спецсимвол', end='')
        print('.')
```

```
kfd$$@dfh
```

```
Пароль ненадежен. Рекомендации: увеличить число символов - 9, добавить 1 з
аглавную букву, добавить 1 цифру.
```

§ 3.10 Байтовые строки

Новый тип в Python – это **байтовые строки, тип `bytes`** .

> **Байт** – это минимальная единица хранения и передачи информации.
> Последовательность байт представляет собой какую-либо информацию (текст, картинку, мелодию...).

Мы не можем, например, передавать по сети строки в виде юникодных символов. Нам нужно преобразовывать строки сначала в байты. То же самое, если мы хотим сохранить строку на диск, то диск работает с байтами. Нам нужно преобразовать наши юникодные символы в байты для этой операции.

В Python для того, чтобы объявить байтовую строку используется литерал `b` .

Пример. Объявим байтовую строку (используем литерал `b`) и передадим в нее строчку `Hello` .

In [3]:

```
example_bytes = b"Hello"
print(type(example_bytes))
```

```
<class 'bytes'>
```

В данном случае мы получаем **байтовую строку** – она представляет собой последовательность чисел от 0 до 255. У нас получилось это сделать, потому что мы передали в литерал байтовой строки ASCII символы. То есть, символы, которые имеют свой номер в таблице ASCII от 0 до 255 и, соответственно, преобразовываются в байты очень легко. Удостоверимся, что мы действительно получили байтовую строку (проитерируемся по нашей байтовой строке):

In [4]:

```
for item in example_bytes:
    print(item)
```

```
72
101
108
108
111
```

Видим, что наша байтовая строка содержит числа от 0 до 255. Однако, если мы попробуем преобразовать в байты нашу юникодную строку, которая, в данном случае `Привет` , мы получим ошибку.

В [5]:

```
example_bytes = b"Привет"
```

```
  File "<ipython-input-5-bd3845541666>", line 1
    example_bytes = b"Привет"
                    ^
SyntaxError: bytes can only contain ASCII literal characters.
```

Как раз потому, что байты – это числа от 0 до 255, и нам нужно юникодные символы преобразовать в байты. Для этого *используются кодировки*. Самая известная кодировка – это UTF-8. На примере покажем, как обычную строку, в данном случае строку `Привет`, преобразовать в байты, используя кодировку UTF-8. Для этого создадим строковый объект, убедимся, что это строка. А дальше воспользуемся методом `encode()`, который есть у строки.

В [6]:

```
example_string = "Привет"
print(type(example_string))

encoder_string = example_string.encode(encoding = 'utf-8')
print(encoder_string)
print(type(encoder_string))
```

```
<class 'str'>
b'\xd0\x9f\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'
<class 'bytes'>
```

`Encode()` принимает опциональным параметром кодировку, в которую мы хотим закодировать нашу строку. В данном случае UTF-8. В принципе, это кодировка по умолчанию, поэтому мы могли бы просто использовать метод `encode` без каких-то дополнительных аргументов. В результате мы видим, что мы получаем байтовую строку, то есть, последовательность байт.

Python напечатал на экран байтовую строку в виде последовательности шестнадцатиричных символов (`\xd0\x9f\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82`). Вначале мы видим символы `d09f` – это представление большой кириллической буквы "П" в кодировке UTF. Если мы посмотрим в шестнадцатиричном представлении как выглядит буква "П" в кодировке UTF, мы как раз увидим эти шестнадцатиричные символы `d09f`. То же самое с другими буквами.

Чтобы декодировать байты обратно в строку необходимо воспользоваться методом строки `decode()`, который есть у байтов. Вызываем метод `decode()` и получаем обратно UTF строку.

В [7]:

```
decoder_string = encoder_string.decode()
print(decoder_string)
```

```
Привет
```

`Decode()` может принимать параметром кодировку, которую мы будем использовать. Но, так как мы работаем в данном случае с UTF -8, это значение по умолчанию, мы просто пишем `decode()` без каких-

то аргументов.

§ 3.11 Ключевые слова

В стандартном пакете Python 3 существуют **ключевые слова**, которым отведен определенный функционал. Перечислим их, чтобы вы ознакомились с ними и в будущем избежали совпадений в названии своих переменных.

Таблица. Ключевые слова Python

Ключевое слово	Описание
`False`	Показатель ложности для булева типа
`True`	Показатель истинности для булева типа
`None`	"Пустой" объект
`and`	Логический оператор `И`
`with/as`	Менеджер контекста
`assert`	Условие, вызывающее исключение, если условие ложно
`break`	Оператор выхода из цикла
`class`	Тип, состоящий из методов и атрибутов
`continue`	Оператор перехода на следующую итерацию цикла
`def`	Определение функции
`del`	Инструкция удаляет переменные, элементы, ключи, срезы и атрибуты
`elif`	Условный оператор в противном случае-если
`else`	Условный оператор в противном случае
`except`	Оператор перехвата исключения
`finally`	Выполняет инструкцию вне зависимости от исключения
`for`	Оператор цикла `for`
`from`	Оператор импорта из модуля
`global`	Оператор создания доступности обращения к переменной за пределами функции
`if`	Условный оператор если
`import`	Оператор импорта модуля
`in`	Оператор проверки на вхождение
`is`	Оператор проверки ссылаются ли 2 объекта на одно место в памяти
`lambda`	Определение анонимной функции
`nonlocal`	Оператор создание доступности обращения к переменной в объемлющей инструкции
`not`	Логический оператор `не`
`or`	Логический оператор `или`
`pass`	Ничего не выполняющий оператор
`raise`	Оператор вызова исключения
`return`	Оператор возвращения результата
`try`	Выполнить инструкции, перехватив исключения
`while`	Условно-циклический оператор до тех пор

Ключевое слово	Описание
`yield`	Определение функции-генератора

Помимо ключевых слов, в Python 3 присутствуют встроенные функции.

Таблица. Встроенные функции Python

Функция	Описание
`bool(x)`	Преобразование к типу `bool`, использующая стандартную процедуру проверки истинности. Если `x` является ложным или опущен, возвращает значение `False`, в противном случае она возвращает `True`
`bytearray([источник [, кодировка [ошибки]]])`	Преобразование к `bytearray`. `Bytearray` - изменяемая последовательность целых чисел в диапазоне $0 \leq X < 256$. Вызванная без аргументов, возвращает пустой массив байт
`bytes([источник [, кодировка [ошибки]]])`	Возвращает объект типа `bytes`, который является неизменяемой последовательностью целых чисел в диапазоне $0 \leq X < 256$. Аргументы конструктора интерпретируются как для `bytearray()`
`complex([real[, imag]])`	Преобразование к комплексному числу
`dict([object])`	Преобразование к словарю
`float([X])`	Преобразование к числу с плавающей точкой. Если аргумент не указан, возвращается 0.0
`frozenset([последовательность])`	Возвращает неизменяемое множество
`int([object], [основание системы счисления])`	Преобразование к целому числу
`list([object])`	Создает список
`memoryview([object])`	Создает объект `memoryview`
`object()`	Возвращает безликий объект, являющийся базовым для всех объектов
`range([start=0], stop, [step=1])`	Арифметическая прогрессия от `start` до `stop` с шагом `step`
`set([object])`	Создает множество
`slice([start=0], stop, [step=1])`	Объект среза от `start` до `stop` с шагом `step`
`str([object], [кодировка], [ошибки])`	Строковое представление объекта. Использует метод `__str__`
`tuple(obj)`	Преобразование к кортежу
`abs(x)`	Возвращает абсолютную величину (модуль числа)
`all(последовательность)`	Возвращает `True`, если все элементы истинные (или, если последовательность пуста)
`any(последовательность)`	Возвращает `True`, если хотя бы один элемент - истина. Для пустой последовательности возвращает `False`
`ascii(object)`	Как `repr()`, возвращает строку, содержащую представление объекта, но заменяет не-ASCII символы на экранированные последовательности
`bin(x)`	Преобразование целого числа в двоичную строку
`callable(x)`	Возвращает `True` для объекта, поддерживающего вызов (как функции)
`chr(x)`	Возвращает односимвольную строку, код символа которой равен `x`
`classmethod(x)`	Представляет указанную функцию методом класса
`compile(source, lename, mode, ags=0, dont_inherit=False)`	Компиляция в программный код, который впоследствии может выполниться функцией `eval` или `exec`. Строка не должна содержать символов возврата каретки или нулевые байты
`delattr(object, name)`	Удаляет атрибут с именем `name`

Функция	Описание
`dir([object])`	Список имен объекта, а если объект не указан, список имен в текущей локальной области видимости
`divmod(a, b)`	Возвращает частное и остаток от деления `a` на `b`
`enumerate(iterable, start=0)`	Возвращает итератор, при каждом проходе предоставляющем кортеж из номера и соответствующего члена последовательности
`eval(expression, globals=None, locals=None)`	Выполняет строку программного кода
`exec(object[, globals[, locals]])`	Выполняет программный код на Python
`filter(function, iterable)`	Возвращает итератор из тех элементов, для которых `function` возвращает истину
`format(value[,format_spec])`	Форматирование (обычно форматирование строки)
`getattr(object, name ,[default])`	Извлекает атрибут объекта или `default`
`globals()`	Словарь глобальных имен
`hasattr(object, name)`	Имеет ли объект атрибут с именем `name`
`hash(x)`	Возвращает хеш указанного объекта
`help([object])`	Вызов встроенной справочной системы
`hex(x)`	Преобразование целого числа в шестнадцатеричную строку
`id(object)`	Возвращает "адрес" объекта. Это целое число, которое гарантированно будет уникальным и постоянным для данного объекта в течение срока его существования
`input([prompt])`	Возвращает введенную пользователем строку. `Prompt` - подсказка пользователю
`isinstance(object, ClassInfo)`	Истина, если объект является экземпляром `ClassInfo` или его подклассом. Если объект не является объектом данного типа, функция всегда возвращает ложь
`issubclass(класс, ClassInfo)`	Истина, если класс является подклассом `ClassInfo` . Класс считается подклассом себя.
`iter(x)`	Возвращает объект итератора
`len(x)`	Возвращает число элементов в указанном объекте
`locals()`	Словарь локальных имен
`map(function, iterator)`	Итератор, получившийся после применения к каждому элементу последовательности функции `function`
`max(iter, [args ...] * [, key])`	Максимальный элемент последовательности
`min(iter, [args ...] * [, key])`	Минимальный элемент последовательности
`next(x)`	Возвращает следующий элемент итератора
`oct(x)`	Преобразование целого числа в восьмеричную строку
`open(le, mode='r', buering=None, encoding=None, errors=None, newline=None, closefd=True)`	Открывает файл и возвращает соответствующий поток
`ord(c)`	Код символа
`pow(x, y[, r])`	Идентично выражению `( x ** y ) % r`
`reversed(object)`	Итератор из развернутого объекта
`repr(obj)`	Представление объекта
`print([object, ...], *, sep=" ", end='\n', le=sys.stdout)`	Вывод результата на экран

Функция	Описание
`property(fget=None, fset=None, fdel=None, doc=None)`	Возвращает специальный объект дескриптора
`round(X [, N])`	Округление до `N` знаков после запятой
`setattr(объект, имя, значение)`	Устанавливает атрибут объекта
`sorted(iterable[, key][, reverse])`	Отсортированный список
`staticmethod(function)`	Статический метод для функции
`sum(iter, start=0)`	Сумма членов последовательности
`super([тип [, объект или тип]])`	Доступ к родительскому классу
`type(object)`	Возвращает тип объекта
`type(name, bases, dict)`	Возвращает новый экземпляр класса name
`vars([object])`	Словарь из атрибутов объекта. По умолчанию - словарь локальных имен
`zip(*iters)`	Итератор, возвращающий кортежи, состоящие из соответствующих элементов аргументов-последовательностей

§ 3.12 Задачи для самостоятельного выполнения

Задача 1. Питон Пончик, он же Питончик, решил вспомнить старые времена, когда было модно писать текстовые сообщения, чередуя заглавные и маленькие буквы. Он захотел написать программу, которая будет делать с любой входной строкой аналогичное действие. Ваша задача: помочь Питончику с учетом того, что пробелы и знаки препинания не должны портить чередование регистра символов (они в этом процессе не учитываются, но возвращаются в итоговой строке).

Номер теста	Входная строка	Результат работы программы
Тест 1	Было сТрашно, но он всЁ-Таки проДолжал идти.	БыЛо СтРаШнО, нО оН вСё-ТаКи ПрОдОлЖаЛ иДтИ.
Тест 2	Ты, Питончик, конечно, Крут! Но есть ещё Круче!	Ты, ПиТоНчИк, КоНеЧнО, кРуТ! нО еСтЬ еЩё КрУчЕ!
Тест 3	а, Б, В, г, Д - первые буквы алФАвита.	А, б, В, г, Д - пЕрВыЕ бУкВы АлФаВиТа.

Задача 2. Питон Пончик, он же Питончик, считает, что текст, записанный в скобках, можно не читать (как вот тут, например). Помогите ленивому Питончику укоротить время чтения: напишите программу, которая в тексте будет удалять все, что находится внутри скобок и сами эти скобки, возвращая очищенный текст (скобки могут быть вложенными). При разработке алгоритма решения нужно учесть, что самая последняя открывающая скобка должна обязательно иметь после себя закрывающую.

Номер теста	Входная строка	Результат работы программы
Тест 1	Падал(лишнее (лишнее) лишнее) прошлогодний снег (лишнее)	Падал прошлогодний снег
Тест 2	(лишнее(лишнее))Падал прошлогодний (лишнее(лишнее(лишнее)))снег	Падал прошлогодний снег
Тест 3	Падал (лишнее) прошлогодний (лишнее) снег (лишнее)	Падал прошлогодний снег

Задача 3. Питон Пончик, он же Питончик, выполняя задание по программированию линейных алгоритмов на PascalABC, где надо было вычислить значение сложного математического выражения получил от компилятора сообщение о неверной расстановке скобок. Он решил написать такую же программу, но немного усложнив условие: проверить, правильно ли во входной строке расставлены квадратные `[]`, круглые `()` и фигурные скобки `{}` (т. е. находится ли справа от каждой открывающей

скобки соответствующая ей закрывающая скобка, а слева от каждой закрывающей — соответствующая ей закрывающая). Ответом должны служить слова «Верно» или «Неверно», причем если скобки расставлены неправильно необходимо вывести номер позиции, в которой раположена первая ошибочная скобка, или, если закрывающих скобок не хватает, вывести "Мало скобок". Помогите Питончику.

Номер теста	Входная строка	Результат работы программы
Тест 1	`((((rlbj)twn}a(we)isr)bwllui`	`Неверно. Мало скобок`
Тест 2	`m(yg({[p{yfrw}n]dvi}qrr)ev)h`	`Верно`
Тест 3	`xe[k){{m{{db{imad}nd}v}ri}c]hm`	`Неверно. Позиция 5`
Тест 4	`[({(rq(vcwje)a)mk}yyl)mux]iz`	`Верно`

Задача 4. Питон Пончик, он же Питончик, пишет своей возлюбленной нежное письмо на русском языке. Чтобы никто, кроме нее, не смог понять глубокий смысл его послания, он решил зашифровать текст, выполнив циклическую замену каждой буквы на букву того же регистра, расположенную в алфавите на k-й позиции ($0 < k < 10$) после шифруемой буквы (например, для $k = 2$ «А» перейдет в «В», «а» — в «в», «Б» — в «Г», «я» — в «б» и т. д.). Букву «ё» в алфавите Питончик решил не учитывать, знаки препинания и пробелы он менять не стал. Помогите Питончику автоматизировать шифрование его писем.

Номер теста	Входная строка	Значение k	Результат работы программы
Тест 1	Нельзя ничего сказать о глубине лужи, пока не попадешь в нее.	2	Пзнюйб пкщзер умвйвфю р енхгкпз нхик, срмв пз срсвжзъю д пзз.
Тест 2	Чтобы одно очистить, нужно другое запачкать.	6	Эшфзб фкуф фэочшошв, ущмуф кцщйфл нжхжэржшв.
Тест 3	Кто может - делает, кто не может - учит, кто не может учить - управляет.	7	Сщх ухнмщ - лмтзмщ, сщх фм ухнмщ - ъюпщ, сщх фм ухнмщ ъюпщг - ъцчзйтжмщ.

Задача 5. Питон Пончик, он же Питончик, пишет своей возлюбленной нежные письма на русском языке, выполняя для шифрования циклическую замену каждой буквы на букву того же регистра, расположенную в алфавите на k-й позиции ($0 < k < 10$) после шифруемой буквы (например, для $k = 2$ «А» перейдет в «В», «а» — в «в», «Б» — в «Г», «я» — в «б» и т. д.). Букву «ё» в алфавите Питончик не учитывает, знаки препинания и пробелы — не менет. Помогите возлюбленной Питончика расшифровать его письма.

Номер теста	Входная строка	k	Результат работы программы
Тест 1	Рфйкж чфзбшое хцоуотждш рцщшфп фзфцфш, ичл чтбиждшче.	6	Когда события принимают крутой оборот, все смываются.
Тест 2	Щъцръ пичняиъиъд чрщдфц, тит к лцуцкы чшрэцмзъ щкнорн фгщур.	8	Стоит запечатать письмо, как в голову приходят свежие мысли.
Тест 3	Зцк зкрнпнк учпхачнд йкрегчцд фу уэнжпк.	5	Все великие открытия делаются по ошибке.

Задание 6. Питон Пончик, он же Питончик, узнал, что слова, которые читаются в обе стороны одинаково, называются *палиндромы*. Он решил написать программу, которая во входном тексте будет обнаруживать слова-палиндромы (без учета регистра букв), но что-то у него не все получается. Помогите Питончику.

Номер теста	Входная строка	Результат работы программы
Тест 1	Казак сказал, что поп шиш получит.	Казак, поп, шиш

Номер теста	Входная строка	Результат работы программы
Тест 2	Мадам Алла, курившая табак, зашла в кабак.	Мадам, Алла, кабак
Тест 3	Ветер с моря дул.	Нет палиндромов

Задание 7. Питон Пончик, он же Питончик, узнал, что слова, которые читаются в обе стороны одинаково, называются *палиндромы*. Кроме того, есть целые предложения-палиндромы. Он решил написать программу, которая проверяет, является ли введенное предложение палиндромом, но что-то у него не все получается: не справляется регистрами букв, которые не учитываются, и с буквой `ё`, которую можно и как `е` использовать. Помогите Питончику.

Номер теста	Входная строка	Результат работы программы
Тест 1	Юлю лис укусил.	Да
Тест 2	Леша на полке клопа нашел.	Да
Тест 3	А лис, он умён — крыса сыр к нему носила.	Да

Задача 8. Питон Пончик, он же Питончик, пишет своей возлюбленной нежное. Чтобы никто, кроме нее, не смог понять глубокий смысл его послания, он решил зашифровать текст, выполняя зеркальное отрожение каждого слова относительно середины. Питончик решил, что в его письмах требование русского языка о том, что предложение должно начинаться с заглавной буквы, должно соблюдаться. Помогите Питончику автоматизировать шифрование его писем.

Номер теста	Входная строка	Результат работы программы
Тест 1	Привет! Как твои дела? Очень скучаю.	Тевирп! Как иовт алед? Ьнечо юачукс.
Тест 2	Для укрепления наших отношений предлагаю сходить в ЗАГС.	Ялд яинелперку хишан йинешонто юагалдерп ьтидохс в САГЗ.
Тест 3	Какую звезду достать с неба?	Юукак удзевз ьтатсод с абен?

Задача 9. Питон Пончик, он же Питончик, пишет своей возлюбленной нежное письмо на русском языке. Чтобы никто, кроме нее, не смог понять глубокий смысл его послания, он решил зашифровать текст, выполняя зеркальное отрожение каждого слова относительно середины. Помогите возлюбленной Питончика расшифровать его письма.

Номер теста	Входная строка	Результат работы программы
Тест 1	Тевирп! Как иовт алед? Ьнечо юачукс.	Привет! Как твои дела? Очень скучаю.
Тест 2	Ялд яинелперку хишан йинешонто юагалдерп ьтидохс в САГЗ.	Для укрепления наших отношений предлагаю сходить в ЗАГС.
Тест 3	Юукак удзевз ьтатсод с абен?	Какую звезду достать с неба?

Задача 10. Питон Пончик, он же Питончик, уже большой змеёныш: он в школе уже прошел операции сложения, вычитания, умножения и деления, а также знает, что такое вещественные числа. По математике ему задали решить много примеров, но ему очень хочется поиграть в любимую игру "Змейка". А его одноклассник — удав Чик (по кличке Удавчик) написал программу, которая сама по введенной строке типа `a operand b` находит ответ и выводит его в виде `a operand b = result`. Помогите Питончику написать программу решения `N` задач, как у Удавчика.

Номер теста	Входная строка	Результат работы программы

Номер теста	Входная строка	Результат работы программы
Тест 1	50.6 / 2	50.6 / 2 = 25.3
Тест 2	2.69 / 0	2.69 / 0 = Деление на ноль
Тест 3	0.0 / 0	0.0 / 0 = Неопределенность
Тест 4	45.5 - 96.3	45.5 - 96.3 = -50.8