Регулярные выражения – это формальный язык для работы со строками. Он позволяет находить подстроки по специально заданным шаблонам и проводить с ними различные манипуляции. В Python для работы с регулярными выражениями есть специальный модуль `re`.

https://docs.python.org/3/library/re.html

Регулярные выражения могуть быть довольно запутанными. Чтобы проверить, правильно ли вы составили регулярку, можно воспользоваться сайтом https://regex101.com/ (выберите Python flavor).

Само по себе регулярное выражение – это строка, задающая некоторый шаблон. Модуль `re` содержит несколько функций, которые позволяют проводить операции над строками с помощью этих шаблонов. Например, функция `findall` позволяет найти все подстроки, соответствующие шаблону. Она принимает на вход два аргумента: регулярное выражение и обрабатываемую строку, а возвращает список подстрок.

Регулярное выражение может задавать подстроку в точности:

```
import re
text = "aaabdbaacaabcd"
substring = "aa"

re.findall(substring, text)
```

...а может быть более общим. Для этого есть специальный символ-джокер `.` и различные управляющие символы:

`+` – один или больше предшествующих символов;

```
re.findall("a.", text)
```

`*` – ноль или больше предшествующих символов;

```
re.findall("a*b", text)
```

`?` – ноль или один предшествующий символ;

```
re.findall("a?b", text)
```

`[]` – любой из перечисленных символов;

```
re.findall("[ab]+", text)
```

`[^]` – любой не из перечисленных символов;

```
re.findall("[^ab]+", text)
```

`[-]` – любой из последовательности символов: каждый символ имеет порядковый номер в Unicode, они сортируются по этому номеру.

```
re.findall("[a-z]", text)
```

Если необходимо найти в тексте сами эти символы, то их нужно экранировать с помощью обратного слеша `\`.

```
text_2 = "abaca. abccd. cdabac. a."
re.findall(r"[a-z]\.", text_2)
```

Поскольку Python также использует обратный слеш для экранирования, рекомендуется оформлять регулярные выражения как r-строки, чтобы одно не путалось с другим.

Стоит заметить, что с регулярными выражениями можно работать двумя путями: они могут быть строками или скомпилированными регулярными выражениями, для которых функции `re` доступны в виде методов:

```
text_2 = "abaca. abccd. cdabac. a."
regex = re.compile(r"[a-z]\.")
regex.findall(text_2)
```

Задание для выполнения в классе: напишите регулярное выражение, которое найдёт в следующем тексте все двузначные числа от `00` до `59`.

```
text = "43 89 72 01 34 42 80 12 99 45 34 29 58"
```

Для некоторых классов символов есть специальные обозначения:

`\w` – словесные символы (цифры, буквы и нижнее подчёркивание);
`\d` – цифры;

`\s` – пробельные символы (пробел, перевод строки, табуляция, etc);

`\W, \D, \S` – обратны вышеперечисленным.

`\b` – соответствует границе слова.

```
re.findall(r"\w+", text_2)
```

Задание для выполнения в классе: напишите регулярное выражение, которое найдёт в английском тексте все наречия с суффиксом -ly.

```
text = "He was carefully disguised but captured quickly by police."
```

Другие символы регулярных выражений:

`{}` – позволяют задать количество повторений предыдущего символа.

```
text = "aabbaaabbbaaaabbbaaabb"
re.findall("a{3}", text)
```

```
re.findall("a{1,3}", text) # диапазон
```

```
re.findall("a{,3}", text) # от нуля
```

```
re.findall("a{2,}", text) # до бесконечности
```

`|` – оператор "или".

```
text = "abc123abc32abced331"
re.findall("\d{3}|abc", text)
```

`^` – начало строки, `$` – конец строки.

```
re.findall("^abc", text)
```

```
re.findall(".$", text)
```

В функцию можно передать специальный флаг `re.MULTILINE`, чтобы начало и конец строки также определялись по символу перевода строки.

```
text3 = "abc\n123\nabc\n32\nabced\n331"
re.findall("^.", text3, re.MULTILINE)
```

Операторы `*`, `.` и `?` – жадные. Они захватывают столько текста, сколько могут.

```
text = "<p>some text</p>"
re.findall("<.+>", text)
```

Чтобы изменить их поведение, поставим дополнительный `?`.

```
text = "<p>some text</p>"
re.findall("<.+?>", text)
```

Функция `findall` возвращает список строк. Другие функции из `re` возвращают т.н. Match object – например, функция `match`, которая определяет, соответствует ли начало строки шаблону. Если соответствия не обнаружилось, такие функции возвращают `None`.

```
m = re.match(r"\d\d", "12abcdef")
print(m.start(), m.end()) # начало и конец
print(m.span()) # начало и конец в одном кортеже
print(m.group()) # совпавший фрагмент строки
print(m.string) # исходная строка
print(m.re) # исходное регулярное выражение
```

Функция `search` сканирует строку, пока не найдётся подходящая подстрока.

```
re.search(r"\d", "abc1bcd32") # первый встретившийся
```

Функция `fullmatch` позволяет понять, соответствует ли строка шаблону полностью.

```
re.fullmatch(r"\w+", "abc1bcd32")
```

Функция `finditer` позволяет итеративно находить все подходящие подстроки.

```
for m in re.finditer(r"\d", "a1bcd34ef1"):
    print(m)
```

Функция `sub` позволяет заменить подстроку какой-либо другой строкой.

```
text = "abc   abd\tabc  a bc      a"
res = re.sub(r"\s+", " ", text)
print(res)
```

Функция `split` позволяет разбить строку, используя подходящие подстроки как разграничители.

```
text = "ab-abb-cbbd abb-bc_bb abbc bbc"
re.split("[ _-]", text)
```

Задание для выполнения в классе: напишите регулярное выражение, которое поможет отобрать из данного списка все слова с ударением на втором слоге (ударение обозначено цифрой `1` после буквы, обозначающей ударный гласный звук).

```
words = [
    "сто1лик",
    "уда1чно",
    "завуали1ровав",
    "изверже1ние",
    "взима1в",
    "репи1тер",
    "нормализова1в",
    "бульдо1г"
]
```

Круглыми скобками можно группировать части регулярного выражения. Каждая часть в круглых скобках соответствует группе в получившемся Match object. Эти группы можно получать по индексу (нулевой индекс соответствует всей найденной подстроке):

```
text = "123abc4bac3288d"

for m in re.finditer(r"(\d)([a-z])", text):
    print(m[0], m[1], m[2])
```

Если вы не хотите, чтобы круглые скобки создавали группу, добавьте в начало `?:` .

```
text = "123abc4bac3288d"

for m in re.finditer(r"(?:\d)([a-z])", text):
    print(m[0], m[1])
```

К группам можно получать доступ, например, при замене:

```
text = "123abc4bac3288d"

print(re.sub(r"(\d)([a-z])", r"\1!\2", text))
```

Или в том же самом выражении:

```
text = "2a3a123a123a1343b43b43a13b3b"
re.findall(r"(([a-z])\d+\2)", text)
```

```
[('a3a', 'a'), ('a123a', 'a'), ('b43b', 'b'), ('b3b', 'b')]
```

Обратите внимание, что теперь `findall` возвращает список не строк, а кортежей, каждый элемент которых соответствует группе. Если группа будет одна, то это опять будет список строк, но строки будут соответствовать не всему выражению, а группе.

Группам можно присваивать имена:

```
text = "123abc4bac3288d"

for m in re.finditer("(?P<name>\d)([a-z])", text):
    print(m['name'])
```

Задание для выполнения в классе: напишите программу, которая с помощью регулярных выражений ставит вокруг каждой гласной буквы квадратные скобки:

```
text = "был тихий серый вечер. дул ветер, слабый и тёплый. небо было покрыто тучами"
```

Задание для выполнения в классе: напишите программу, которая с помощью регулярных выражений генерирует таблицу успеваемости по тексту:

```
text = 'Оценка ученика по предмету "математика" - 5. По предмету "биология" - 4. По предмету
```

Домашнее задание:

1. Напишите регулярное выражение, описывающее стандартный автомобильный номер (буква – три цифры – две буквы – две или три цифры региона). Учтите, что (а) буквы могут быть только те, которые есть и в латинице, и в кириллице; (б) если в регионе три цифры, то первая – либо 1, либо 7.

2. В созданном вами произносительном словаре найдите слова, в которых встречается три или больше согласных звуков подряд. Составьте частотный рейтинг консонантных кластеров.

Есть такая игра – Regex Golf (https://alf.nu/RegexGolf). Суть – создать как можно более короткое регулярное выражение, которое соответствует (`re.search`) всем словам из одного списка и ни одному слову из другого списка.