Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

В данной лабораторной работе будет рассмотрен более сложный механизм проверки на совпадение с шаблоном, чем рассмотренные ранее — регулярные выражения. Этот механизм поставляется в стандартном модуле **re**, который мы импортируем. Вы определяете строковый шаблон, совпадения для которого вам нужно найти, и строку-источник, в которой следует выполнить поиск. Простой пример использования выглядит так:

```
>>> result = re.match('You', 'Young Frankenstein')
>>> result
<_sre.SRE_Match object; span=(0, 3), match='You'>
```

В этом примере строка 'You' является шаблоном, а 'Young Frankenstein' — источником, строкой, которую вы хотите проверить. Функция match() проверяет, начинается ли источник с шаблона.

Для более сложных проверок нужно скомпилировать шаблон, чтобы ускорить поиск. Далее вы можете выполнить проверку с помощью скомпилированного шаблона:

```
>>> yourpattern = re.compile('You')
>>> result = yourpattern.match('Young Frankenstein')
```

Функция match() — это не единственный способ сравнить шаблон и источник, существует еще несколько методов.

- search() возвращает первое совпадение, если таковое имеется.
- findall() возвращает список всех непересекающихся совпадений, если таковые имеются.
- split() разбивает источник на совпадения с шаблоном и возвращает список всех фрагментов строки.
- sub() принимает аргумент для замены и заменяет все части источника, совпавшие с шаблоном, на значение этого аргумента.

Точное совпадение с помощью функции match()

Начинается ли строка 'Young Frankenstein' со слова 'You'? Рассмотрим пример кода с комментариями:

```
>>> import re
>>> source = 'Young Frankenstein'
>>> m = re.match('You', source)  # функция начинает работать с начала источника
>>> if m:  # функция возвращает объект; делайте это, чтобы увидеть, что совпало
...     print(m.group())
...
You
>>> m = re.match('^You', source) # якорь в начале строки делает то же самое
>>> if m:
...     print(m.group())
...
You
```

Теперь рассмотрим фрагмент текста 'Frank'.

```
>>> m = re.match('Frank', source)
>>> if m:
...     print(m.group())
...
```

В этот раз функция match() не вернула ничего, и оператор if не запустил оператор print. Как я говорил ранее, функция match() работает только в том случае, если шаблон находится в начале источника. Но функция search() ищет шаблон в любом месте источника:

```
>>> m = re.search('Frank', source)
>>> if m:
...       print(m.group())
...
Frank
```

Изменим шаблон:

```
>>> m = re.match('.*Frank', source)
>>> if m:  # match returns an object
...     print(m.group())
...
Young Frank
```

Рассмотрим, как работает наш новый шаблон:

- символ . означает любой символ;
- символ * означает любое количество предыдущих элементов. Если объединить символы .*, они будут означать любое количество символов (даже ноль);
- 'Frank' — это фраза, которую мы хотим найти в любом месте строки.

Функция match() вернула строку, в которой нашлось совпадение с шаблоном .*Frank: 'Young Frank'.

Первое совпадение, найденное с помощью функции search()

Вы можете использовать функцию search(), чтобы найти шаблон 'Frank' в любом месте строки-источника 'Young Frankenstein', не прибегая к использованию символа подстановки .*:

```
>>> m = re.search('Frank', source)
>>> if m:  # функция search возвращает объект
...     print(m.group())
...
Frank
```

Ищем все совпадения с помощью функции findall()

В предыдущих примерах мы искали только одно совпадение. Но что, если вы хотите узнать, сколько раз строка, содержащая один символ n, встречается в строке-источнике?

```
>>> m = re.findall('n', source)
>>> m   # findall returns a list
['n', 'n', 'n', 'n']
>>> print('Found', len(m), 'matches')
Found 4 matches
```

Как насчет строки 'n', за которой следует любой символ?

```
>>> m = re.findall('n.', source)
>>> m
['ng', 'nk', 'ns']
```

Обратите внимание на то, что в совпадения не была записана последняя строка 'n'. Нам нужно сказать, что символ после 'n' является опциональным, с помощью конструкции ?:

```
>>> m = re.findall('n.?', source)
>>> m
['ng', 'nk', 'ns', 'n']
```

Разбиваем совпадения с помощью функции split()

В следующем примере показано, как разбить строку на список с помощью шаблона, а не простой строки (как это делает метод split()):

```
>>> m = re.split('n', source)
>>> m    # функция split возвращает список
['You', 'g Fra', 'ke', 'stei', '']
```

Заменяем совпадения с помощью функции sub()

Этот метод похож на метод replace(), но он ищет совпадения с шаблонами, а не простые строки:

```
>>> m = re.sub('n', '?', source)
>>> m   # sub returns a string
'You?g Fra?ke?stei?'
```

Шаблоны: специальные символы

Многие описания регулярных выражений начинаются с деталей, касающихся того, как их определить. Я считаю, что это ошибка. Язык регулярных выражений не так уж мал сам по себе, слишком много деталей должно вместиться в вашу голову одновременно. Они используют так много знаков препинания, что это выглядит так, будто персонажи мультиков ругаются.

Теперь, когда вы знаете о нужных функциях (match(), search(), findall() и sub()), рассмотрим детали построения регулярных выражений. Создаваемые вами шаблоны подойдут к любой из этих функций.Самые простые знаки вы уже видели.

- Совпадения с любыми неспециальными символами.
- Любой отдельный символ, кроме \n, — это символ ..
- Любое число, включая 0, — это символ *.
- Опциональное значение (0 или 1) — это символ ?.

Специальные символы показаны в табл. 12.1.

Таблица 12.1. Специальные символы

Шаблон	Совпадения
\d	Цифровой символ
\D	Нецифровой символ
\w	Буквенный или цифровой символ или знак подчеркивания
\W	Любой символ, кроме буквенного или цифрового символа или знака подчеркивания
\s	Пробельный символ
\S	Непробельный символ
\b	Граница слова
\B	Не граница слова

Модуль Python string содержит заранее определенные строковые константы, которые мы можем использовать для тестирования. Мы воспользуемся константой printable, которая содержит 100 печатаемых символов ASCII, включая буквы в обоих регистрах, цифры, пробелы и знаки пунктуации:

```
>>> import string
>>> printable = string.printable
>>> len(printable)
100
>>> printable[0:50]
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMN'
>>> printable[50:]
'OPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
```

Какие символы строки printable являются цифрами?

```
>>> re.findall('\d', printable)
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

Какие символы являются цифрами, буквами и нижним подчеркиванием?

```
>>> re.findall('\w', printable)
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b',
'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
'Y', 'Z', '_']
```

Какие символы являются пробелами?

```
>>> re.findall('\s', printable)
[' ', '\t', '\n', '\r', '\x0b', '\x0c']
```

Регулярные выражения не ограничиваются символами ASCII. Шаблон \d совпадет со всем, что в кодировке Unicode считается цифрой, а не только с символами ASCII от 0 до 9. Добавим две буквы в нижнем регистре не из ASCII из FileFormat.info.

В этой проверке мы добавим туда следующие символы:

Ѓ три буквы ASCII;

Ѓ три знака препинания, которые не должны совпасть с шаблоном \w;

Ѓсимвол Unicode LATIN SMALL LETTER E WITH CIRCUMFLEX (\u00ea);
Ѓсимвол Unicode LATIN SMALL LETTER E WITH BREVE (\u0115):

```
>>>x='abc'     +'-/*'    +'\u00ea'+'\u0115'
```

Как и ожидалось, этот шаблон нашел только буквы:

```
>>> re.findall('\w', x)
['a', 'b', 'c', 'к', 'ĕ']
```

Шаблоны: использование спецификаторов

Теперь сделаем строку из знаков препинания, используя основные спецификаторы шаблонов для регулярных выражений, показанные в табл. 12.2.В этой таблице expr и другие слова, выделенные курсивом, означают любое корректное регулярное выражение.

Таблица 12.2. Спецификаторы шаблонов

Шаблон	**Совпадения**
abc	Буквосочетание abc
(expr)	expr
expr1 | expr2	expr1 или expr2
.	Любой символ, кроме \n
^	Начало строки источника
$	Конец строки источника
prev ?	Ноль или одно включение prev
prev *	Ноль или больше включений prev, максимальное количество
prev *?	Ноль или больше включений prev, минимальное количество
prev +	Одно или больше включений prev, максимальное количество
prev +?	Одно или больше включений prev, минимальное количество
prev { m }	m последовательных включений prev
prev { m, n }	От m до n последовательных включений prev, максимальное количество
prev { m, n }?	От m до n последовательных включений prev, минимальное количество
[abc]	a, или b, или c (аналогично a|b|c)
[^abc]	Не (a, или b, или c)
prev (?= next)	prev, если за ним следует next
prev (? ! next)	prev, если за ним не следует next
(?<=prev) next	next, если перед ним находится prev
(?<! prev) next	next, если перед ним не находится prev

У вас могло зарябить в глазах при попытке прочесть эти примеры. Для начала определим строку-источник:

```
>>> source = '''I wish I may, I wish I might
... Have a dish of fish tonight.'''
```

Найдем во всем тексте строку 'wish':

```
>>> re.findall('wish', source)
['wish', 'wish']
```

Далее найдем во всем тексте строки 'wish' или 'fish':

```
>>> re.findall('wish|fish', source)
['wish', 'wish', 'fish']
```

Найдем строку 'wish' в начале текста:

```
>>> re.findall('^wish', source)
[]
```

Найдем строку 'I wish' в начале текста:

```
>>> re.findall('^I wish', source)
['I wish']
```

Найдем строку 'fish' в конце текста:

```
>>> re.findall('fish$', source)
[]
```

Наконец, найдем строку 'fish tonight.$' в конце текста:

```
>>> re.findall('fish tonight.$', source)
['fish tonight.']
```

Символы ^ и $ называются якорями: с помощью якоря ^ выполняется поиск в начале строки, а с помощью якоря $ — в конце. Сочетание .$ совпадает с любым символом в конце строки, включая точку, поэтому выражение сработало. Для обеспечения большей точности нужно создать управляющую последовательность, чтобы найти именно точку:

```
>>> re.findall('fish tonight\.$', source)
['fish tonight.']
```

Начнем с поиска символов w или f, за которым следует буквосочетание ish:

```
>>> re.findall('[wf]ish', source)
['wish', 'wish', 'fish']
```

Найдем одно или несколько сочетаний символов w, s и h:

```
>>> re.findall('[wsh]+', source)
['w', 'sh', 'w', 'sh', 'h', 'sh', 'sh', 'h']
```

Найдем сочетание ght, за которым следует любой символ, кроме буквенного или цифрового символа или знака подчеркивания:

```
>>> re.findall('ght\W', source)
['ght\n', 'ght.']
```

Найдем символ I, за которым следует сочетание wish:

```
>>> re.findall('I (?=wish)', source)
['I ', 'I ']
```

И наконец, сочетание wish, перед которым находится I:

```
>>> re.findall('(?<=I) wish', source)
[' wish', ' wish']
```

Существует несколько ситуаций, в которых правила шаблонов регулярных выражений конфликтуют с правилами для строк Python. Следующий шаблон должен совпасть с любым словом, которое начинается с fish:

```
>>> re.findall('\bfish', source)
[]
```

Почему этого не произошло? Как мы говорили в лабораторной работе 3, Python использует специальные управляющие последовательности для строк. Например, \b для строки означает «возврат на шаг», но в мини-языке регулярных выражений эта последовательность означает начало слова. Избегайте случайного применения

управляющих последовательностей, используя неформатированные строки Python, когда определяете строку регулярного выражения. Всегда размещайте символ r перед строкой шаблона регулярного выражения, и управляющие последовательности Python будут отключены, как показано здесь:

```
>>> re.findall(r'\bfish', source)
['fish']
```

Шаблоны: указываем способ вывода совпадения

При использовании функций match() или search() все совпадения можно получить из объекта результата m, вызвав функцию m.group(). Если вы заключите шаблон в круглые скобки, совпадения будут сохранены в отдельную группу и кортеж, состоящий из них, окажется доступен благодаря вызову m.groups(), как показано здесь:

```
>>> m = re.search(r'(. dish\b).*(\bfish)', source)
>>> m.group()
'a dish of fish'
>>> m.groups()
('a dish', 'fish')
```

Если вы используете этот шаблон (?P< name > expr), он совпадет с выражением expr, сохраняя совпадение в группе name:

```
>>> m = re.search(r'(?P<DISH>. dish\b).*(?P<FISH>\bfish)', source)
>>> m.group()
'a dish of fish'
>>> m.groups()
('a dish', 'fish')
>>> m.group('DISH')
'a dish'
>>> m.group('FISH')
'fish'
```

Контрольные вопросы

1. Что такое регулярные выражения? Для чего они применяются?
2. Соответствие символов.
3. Метасимволы, их значение и применение.
4. Использование регулярных выражений.
5. Компиляция регулярных выражений.
6. Функции для работы с регулярными выражениями.

Список литературы, рекомендуемый к использованию по данной теме:

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