Python

Урок 4. Системное программирование

Регулярные выражения

Регулярные выражения являются мощным языком для проверки совпадения строки с шаблоном. В Python для регулярных выражений используется модуль "re".

В Python поиск по регулярному выражению обычно записывается как:

```
match = re.search(pat, str)
```

Метод re.search() принимает шаблон регулярного выражения и строку, и выполняет поиск шаблона по этой строке. Если шаблон в строке найден — search() возвращает объект match или None в противном случае. Таким образом, поиск, как правило, идет сразу после условного оператора, чтобы проверить, был ли поиск удачен, как показано в следующем примере, который ищет шаблон 'word: ", после которого удут 3 буквы (подробности см. ниже):

```
str = 'an example word:cat!!'
match = re.search(r'word:\w\w\w', str)
# If-statement after search() tests if it succeeded
if match:
    print 'found', match.group() ## 'found word:cat'
else:
    print 'did not find'
```

Код match = re.search(pat, str) сохраняет результат поиска в переменной с именем "match". Затем условный оператор проверяет — если match истинен — поиск удался и match.group() содержит соответствующий текст ('word:cat' в нашем случае). В противном случае, если match ложен (равняется None, если быть точным), то поиск не удался, и в строке нет ни одного совпадения с шаблоном.

'r' в начале строки-шаблона обозначает в Питоне "сырые" строки, в которых обратные слэши хранятся без изменений. Такие строки очень удобны для регулярных выражений. Рекомендуется, чтобы вы всегда создавали свои регулярные выражения с использованием сырых строк.

Базовые шаблоны

Сила регулярных выражений в том, что они могут задавать шаблоны, а не только фиксированные символы. Вот самые основные шаблоны:

```
a, X, 9,
```

Это самые обычные символы, обозначающие сами себя.

Также существуют метасимволы, которые не обозначают сами себя, потому что в них вкладывается специальный смысл:

```
. ^ $ * + ? { } [ ] \ | ( )
```

Шаблон	Эквивалент	Описание
.		обозначает любой символ, кроме символа перевода строки '\n'
\w	[A-Za-z0-9_]	соответствует любой букве, цифре или знаку подчеркивания
\W	[^a-zA-Z0-9_]	соответствует любому символу, кроме букв, цифр и знаков подчеркивания
\s	[\t\n\r\f\v]	соответствует одиночному "пробельному" символу — пробел, новая строка, возврат каретки, табуляция
\S	[^ \t\n\r\f\v]	соответствует любому "не-пробельному" символу

Шаблон	Эквивалент	Описание
\t, \n, \r		табуляция, новая строка, перевод каретки
\d	[0-9]	соответствует любой цифре
\D	[^0-9]	соответствует любому нечисловому символу
\b		соответствует пустой строке, но только в начале или в конце слова; под словом понимается последовательность алфавитно-цифровых или символов подчеркивания, таким образом конец слова определяется пробелом или не алфавитно-цифровым символом и не символом подчеркивания; формально \b определяется как граница между символами \w и \W (или наоборот), или между \w и началом/концом строки; например, r'\bfoo\b' соответствует 'foo', 'foo.', '(foo)', 'bar foo bar', но не 'foobar' или 'foo3'.
^, $		^ — начало, $ — конец; начало и окончание строки
\		подавляют "специализированность" символа; так, например, \. означает точку или \\ означает слэш; если вы не уверены, есть ли у символа специальное значение, как например у @, вы можете поставить перед ним \, чтобы быть уверенным, что @ используется просто как символ

Метасимволы повторения

Символ	Описание
*	предыдущий символ может повторяться 0 или более раз; повторений может быть настолько много, насколько это возможно; например, 'ab*' будет соответствовать 'a', 'ab' или 'a', после которой идет любое количество символов 'b'
+	предыдущий символ может повторяться 1 или более раз; например, 'ab+' будет соответствовать 'ab', 'abb', но не просто 'a'
?	предыдущий символ может повторяться 0 или 1 раз; например, 'ab?' будет соответствовать 'a' и 'ab'

Наборы символов

Для указания рабора символов используются квадратные скобки. Например, [abc] соответствует символам "a" или "b" или "c".

Метасимволы теряют свой специальный внутри наборов символов (квадратных скобок). Например, [(+*)] будет соответствовать любому из этих символов — '(', '+', '*', ')'.

Классы символов, такие как \w, \s и т.п., также работают внутри наборов.

Например, шаблон поиска email:

```
match = re.search(r'[\w.-]+@[\w.-]+', str)
```

Вы также можете использовать дефис для указания диапазона символов, таким образом [a-z] будет соответствовать всем строчным буквам. Чтобы использовать тире без указания диапазона, поставьте тире последним, например, [abc-].

Значек ^ в начале квадратных скобок инвертирует его, таким образом [^ab] означает любой символ кроме 'a' или 'b'.

Работа с группами

Возможность работать с группами в регулярных выражениях позволяет выделять из текста нужные части по шаблону.

Например, мы хотим извлечь части email по отдельности — имя пользователя и домен. Для этого, в регулярном выражении обрамим круглыми скобками имя пользователя и домена:

```
r'([\w.-]+)@([\w.-]+)'
```

В этом случае скобки не влияют на соответствие строки шаблону, но они устанавливают логические "группы" внутри этого шаблона. Если поиск шаблона в тексте будет успешен, то match.group(1) будет содержать текст, соответствующий содержимому 1-й скобки слева, а match.group(2) содержать текст, соответствующий содержимому 2-й скобки. match.group() по-прежнему будет содержать весь текст, как обычно.

Обычно при работе с регулярными выражениями удобно придерживаться следующего порядка: сначала написать шаблон для всего искомого фрагмента

текста, а затем добавить круглые скобки для определения групп и извлечения из текста нужных вам частей.

findall

findall() является, пожалуй, самой мощной функцией модуля 're'. Мы использовали re.search(), чтобы найти первое соответствие шаблону. findall() находит *все* соответствия и возвращает их в виде списка строк, каждая из которых представляет одно соответствие.

```
>>> str = 'some text abc@xyz.com foo bar alice-in-
 chains@gmail.com, freddie@queen.co.uk and on and on and on'
>>> emails = re.findall(r'[\w.-]+@[\w.-]+', str)
>>> emails
['abc@xyz.com', 'alice-in-chains@gmail.com',
 'freddie@queen.co.uk']
```

findall и группировка

Если шаблон всключает в себя 2 или более группы скобок, то вместо того, чтобы возвращать список строк, findall() вернет список *кортежей*.

```
>>> emails = re.findall(r'([\w.-]+)@([\w.-]+)', str)
>>> emails
[('abc', 'xyz.com'), ('alice-in-chains', 'gmail.com'),
 ('freddie', 'queen.co.uk')]
```

Дополнение: иногда в вашем шаблоне есть сгруппированные скобками ( ) выражения, но вы не хотите извлекать их содержимое. В этих случаях напишите скобки с ?: в начале, то есть (?:...), и тогда содержимое этой группы не попадет в результат.

findall и файлы

Для поиска шаблона в файле проще всего передать функции findall() весь файл целиком, чем считывать из файла построчно, а затем передавать полученные строки в findall():

```
# Откроем файл
f = open('test.txt', 'r')
# Передедадим содержимое файла в findall();
# она вернет список всех найденных строк
strings = re.findall(r'pattern', f.read())
```

Флаги

Флаги компиляции позволяют изменять поведение регулярных выражений. Флаги доступны в модуле под двумя именами: длинным, таким как IGNORECASE и коротким, в однобуквенной форме, таким как I. Несколько флагов могут быть заданы в форме двоичного ИЛИ; например re.I | re.M устанавливает флаги I и M.

re.**I**
re.**IGNORECASE**
: Выполняет сравнение без учета регистра; например, [A-Z] будет также соответствовать и строчным буквам.

re.**M**
re.**MULTILINE**
: Если этот флаг указан, спецсимвол ^ означает начало всей строки и также начало каждой строчки (непосредственно следующее после каждого символа новой строки); и символ $ означает конец строчки (непосредственно перед каждым символом новой строки).
По умолчанию, ^ означает только начало всей строки, а $ только коней всей строки.

re.**S**
re.**DOTALL**
: Если уставновлен этот флаг специальный символ '.' соответсвует всем символам, включая символ конца строки \n.

re.**X**
re.**VERBOSE**
: Включает многословные (подробные) регулярные выражения, которые могут быть организованы более ясно и понятно. Если указан этот флаг, пробелы в строке регулярного выражения игнорируется, кроме случаев, когда они имеются в классе символов или им предшествует неэкранированный бэкслеш; это позволяет вам организовать регулярные выражения более ясным образом. Этот флаг также позволяет помещать в регулярные выражения комментарии, начинающиеся с '#', которые будут игнорироваться.

```
a = re.compile(r"""\d +  # the integral part
                   \.    # the decimal point
                   \d *  # some fractional digits""", re.X)
b = re.compile(r"\d+\.\d*")
```

Отладка регулярных выражений

Часто в регулярных выражениях всего в нескольких символах "упаковано" очень много смысла. И этот смысл настолько спрессован, что вы можете потратить много времени на отладку своих шаблонов. Настройте свою среду выполнения таким образом, чтобы вы могли запускать шаблон и легко выводить найденные соответствия, например, запуская поиск совпадений шаблона на маленьком тексте и выводя результат функции findall(). Если шаблон не совпадает ни с чем, попробуйте ослабить ваш шаблон, удалив некоторые его части. Так вы получите много соответствий. Как только соответствий станет слишком много, то вы можете постепенно начать делать шаблон строже, чтобы получить только то, что вам нужно.

Жадные и не жадные

Предположим, у вас есть текст с такими тегами:

```
<b>foo</b> and <i>bar</i>
```

Предположим, что вы пытаетесь найти каждый тег с шаблоном '(<.*>)' — что будет соответствовать ему в первую очередь?

Результат будет немного удивительным, но "жадный" характер выражения .* заставит его соответствовать всей стоке ' foo and bar'. Проблема в том, что .* заходит так далеко, насколько это возможно, вместо того, чтобы останавливаться на первой > (то есть выражение "жадное").

Чтобы сделать регулярное выражение не жадным, нужно добавить в конец выражения знак вопроса '?', то есть .*? или .+?. Теперь оно остановится так скоро, как сможет. Таким образом, шаблон '(<.*?>)' найдет '' как первое совпадение, '' как второе, и так далее, находя пары <..> в каждом шаге. Стиль написания подобных регулярный выражений, как правило, таков, что поисле конструкции .*? сразу идет конкретный маркер (в нашем случае >), который показывает, до каких пор будет работать выражение .*?.

Замена

Функция re.**sub**(*pattern*, *replacement*, *str*, *count=0*, *flags=0*) ищет все вхождения шаблона в заданной строке и заменяет их. Строка замены может включать '\1', '\2'. которые ссылаются на текст из group(1), group(2) и так далее из исходного текста.

В данном примере мы найдем все email адреса и заменим их таким образом, чтобы сохранить имя пользователя (\1), но заменить домен на supermail.com:

```
str = 'purple alice@google.com, blah monkey bob@abc.com blah
 dishwasher'
## re.sub(pat, replacement, str) -- returns new string with all
 replacements,
## \1 is group(1), \2 group(2) in the replacement
print re.sub(r'([\w\.-]+)@([\w\.-]+)', r'\1@yo-yo-dyne.com',
 str)
## purple alice@yo-yo-dyne.com, blah monkey bob@yo-yo-dyne.com
 blah dishwasher
```

Документация по модулю 're':
http://docs.python.org/3/library/re.html

Компиляция регулярных выражений

Если у вас есть регулярное выражение, которое вы хотите использовать многократно — его можно скомпилировать при помощи метода compile():

```
code_chars = re.compile(r'[a-zA-Z0-9_ -]+')
if code_chars.match(q):
    for code in code_chars.findall(q):
        print( re.sub(r'[- ]', '', code) )
```

Файловая система — os, os.path, shutil

Модулы 'os' и 'os.path' включают множество функций для работы с файловой системой. Модуль 'shutil' может копировать файлы.

os.**listdir**(*dir*)
: Возвращает список имен файлов в заданной директории (не включая . и ..). Имена файлов просто названия файлов в директории, не абсолютные пути к ним.

os.**mkdir**(*dir_path*)
: Создает одну директорию.

os.**makedirs**(*dir_path*)
Создает директории в заданном пути.

os.**rmdir**(*path*)
Удаляет директорию. Работает только для пустых директорий. Чтобы удалить директорию со всем ее содержимым shutil.rmtree().

os.path.**join**(*path1*[, *path2*[, ...]])
Объединяет пути к файлу. Например, удобно объединить этой функцией dir и filename из прошлого примера, чтобы получить путь к файлу.

os.path.**abspath**(*path*)
Получает путь, возвращает абсолютный путь, например, /home/vasily/spam/eggs.html

os.path.**dirname**(*path*)
Возвращает название директории из заданного пути. Например, vasily/spam/eggs.html вернет vasily/spam.

os.path.**basename**(*path*)
Возвращает название файла. Например, vasily/spam/eggs.html вернет eggs.html.

os.path.**exists**(*path*)
Истинно, если путь path существует.

shutil.**copy**(*source-path*, *dest-path*)
Копирует файл (конечная директория должна существовать)

shutil.**rmtree**(*path*)
Удаляет дерево директории целиком.

Документация по модулям:
http://docs.python.org/3/library/os.html
http://docs.python.org/3/library/os.path.html
http://docs.python.org/3/library/shutil.html

urllib

urllib.request

Этот модуль определяет функции и классы для работы с URL. Модуль делает работу с url похожей на работу с файлом, который вы можете прочесть.

ufile = urllib.request.**urlopen**(*url*)
Возвращает файлоподобный объект для данного url.

text = ufile.**read**()
Может читать из открытого файла. Как и в случае с обычным файлом, readlines() и т.п. также работает. (Обратите внимание — возвращается байт-строка. Так происходит потому, что нет способа для автоматического определения кодировки потока байт, получаемого от HTTP-сервера. Для перекодировки используйте .decode('utf-8') или .decode('cp1251'):

```
>>> import urllib.request
>>> f = urllib.request.urlopen('http://www.python.org/')
>>> print(f.read(100).decode('utf-8'))
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtm
```

info = ufile.**info**()
Метаинформация о запросе. info.gettype() возвращает mime time, напр. 'text/html'.

baseurl = ufile.**geturl**()
Возвращает базовый url запроса, который может отличаться от оригинального из-за редиректов.

code = ufile.**getcode**()
Возвращает код ответа сервера. 200 — Ок.

urllib.request.**urlretrieve**(*url*, *file_name*)
Скачивает url и сохраняет в файл file_name.

Также для скачивания можно применять такой метод:

```
with urllib.request.urlopen(url) as response, open(file_name,
 'wb') as out_file:
    shutil.copyfileobj(response, out_file)
```

urllib.parse

Модуль urllib работу url-выборки —. Модуль *urlparse* может разбирать и собирать url.

urllib.parse.**urlencode**(*query*)

Преобразует словать или списком кортежей из 2х элементов в строку, состоящую из пар ключ=значение, разделенных символом & и закодированную как URL (англ. «percent-encoding»).

urllib.parse.**urlparse**(*urlstring*)

Разделяет URL на 6 частей: scheme://netloc/path;parameters?query#fragment