RegEx (Регулярные выражения)

Регулярные выражения - это очень мощный инструмент для поиска соответствий строк. В качестве формулы поиска используются специальные метасимволы, которые позволяют описать правило, по которому мы можем определить - соответствует какая-то фраза, слово или текст критерию нашего поиска.

Обычно поиск подстроки в тексте в Python выполняется очень просто - обычной проверкой if in.

```
if "o Worl" in "Hello World":
    print( "YES" )
```

В отличие от обычного поиска подстроки, регулярные выражения позволяют определить, например, есть ли в строке подстрока, которая начинается с цифры (причем с любой цифры от 0 до 5), потом где-то в середине строки присутствует слово *IK* или *FK*, и заканчиваться строка должна одним из символов в данном списке ["a", "z", "b"].

Увидеть в действии, что можно делать с регулярными выражениями, можно в следующем примере:

```
import re

p = re.compile("^[1-5]+.*[_FK_|_IK_]+.*[azb]+\Z")

s_1 = "5_blahblag_IK_blahblag_z"
s_2 = "6_blahblag_IK_blahblag_z"
s_3 = "2_blahblag_FK_blahblag_a"


print(p.match(s_1))# <_sre.SRE_Match object; span=(0, 24), match='5_blahblag_IK_blahblag_z'>
print(p.match(s_2))# None т.к. вначале уже символ 6
print(p.match(s_3))# <_sre.SRE_Match object; span=(0, 24), match='2_blahblag_FK_blahblag_a'>
```

Разберем более подробно, что здесь написано.

```
import re

p = re.compile("^[1-5]+.*[_FK_|_IK_]+.*[azb]+\Z")
```

Внутри compile мы пишем формулу, по которой мы будем производить поиск. Данная формула состоит из набора символов, каждый из которых имеет свою цель. Например **[1-5a-z]** говорит о том, что это может быть абсолютно любой символ от 1 до 5 либо от буквы a до z, а символ .* означает что это может быть абсолютно любой символ в любом количестве, т.е. asdf234 и 19-4_323ii будут соответствовать этому символу.

```
^[1-5]+.*[_FK_|_IK_]+.*[azb]+\Z
```

- **^** обозначает начало строки, за которым следует
- **[1-5]+** В квадратных скобках мы перечисляем символы, которые мы ожидаем встретить. Любые метасимволы внутри [] теряют свои свойства и ищутся в строке как и обычные символы, например [\d] произведет поиск символов \ и d.
- Знак **+** говорит о том, что любой символ от 1 до 5 (1,2,3,4,5), должен встретиться в этой последовательности как минимум 1 раз. Т.е. наш шаблон будет найден в строке, у которой в начале первый символ является цифрой от 1 до 5.
- **.*** означает что далее следуют абсолютно любые символы в любом количестве
- **[_FK_|_IK_]+** значит что далее должно встретиться одно из последовательностей символов *FK*, или *IK* как минимум один раз
- **[azb]+** значит что после любого числа любых символов .* должен встретиться один из символов a z или b как минимум один (и может быть более) раз
- После чего мы ставим **\Z**, указывая на конец строки

Операция **p.match(s_1**) говорит о том - соответствует ли наша формула строке s_1. Если функция возвращает None, значит совпадение не найдено. Если возвращает набор странных символов, то это указатель на объект класса _sre.SRE_Match, иными словами - совпадение найдено.

В интернете очень большое количество примеров, по которым можно быстро научиться использовать регулярные выражения. Я рекомендую обращать внимание на статьи вроде **RegEx Cheat Sheet** (https://www.rexegg.com/regex-quickstart.html).

Разберем более подробно на простых примерах.

Проверить, соответствует ли формула поиска заданной строке. В данном случае мы прямо проверяем символы **xyz = xyz**.

```
text = "xyz"
formula = re.compile("xyz")
print ( formula.match(text) ) #  в ы в е д е т  <re.Match object; span=(0, 3), match='xyz'>
```

Если print выведет нам None - значит строки не сходятся. Однако, если результат отличается - значит формула и строка сошлись. Попробуем изменить текст строки.

```
text = "xyzabcde"
formula = re.compile("xyz")
print ( formula.match(text) ) #  в ы в е д е т  <re.Match object; span=(0, 3), match='xyz'>
```

В данном случае, результат вывода print будет таким же. Потому что формула xyz смогла найти комбинацию этих символов в строке xyzabcde.

Далее, мы можем начать добавлять условия в формулу, чтобы проверить - совпадает ли она со строкой или нет. Фактически сложность формулы это просто набор простых условий. Чем больше условий, тем сложнее кажется формула, однако она является лишь набором простых специальных символов.

Коллекция символов

Рассмотрим примеры коллекций символов, которые пишутся в квадратных скобках. Суть их в следующем - мы можем в квадратных скобках написать любое количество любых символов в произвольном порядке, и вся эта штуковина будет представлять из себя один любой символ из этой коллекции. Например **[asdbv33]** является одновременно и a и s и любым другим символом из этой коллекции. Где это применяется? - допустим мы знаем что наше слово, которое мы ищем, начинается с буквы x, заканчивается на букву z, а между ними может быть любой символ латинского алфавита либо любая цифра.

```
text1 = "xgz"
text2 = "x6z"
text3 = "xuz"
formula = re.compile("x[0-9a-zA-B]z") # 0-9 значит 0,1,2,3,4,5,6,7,8,9. То же самое с буквами
print ( formula.match(text1) ) # Ура - формула и текст совпадает!
```

[0-9a-zA-B] - означает любой символ (либо цифра от 0 до 9, либо маленькая или большая буква от a до Z)

Мы можем упростить эту формулу, написав x.z. Точка . означает абсолютно любой символ, даже такой как **$, &, _, ~** - в общем любой вообще из существующих. Поэтому следует дважды подумать, не найдет ли такая формула ненужные варианты, такие как x$z или x z.

Последовательное число символов

С помощью специальных знаков мы можем определить, сколько раз тот или иной символ может встретиться в строке. Например **xy*z** (звездочка) означает что между x и z может встретиться любое количество символов y. Например, данная формула будет соответствовать строкам **xz**, **xyz**, **xyyyyyyz**.

Знак + означает, что символ может встретиться любое количество раз подряд, но, он должен присутствовать в тексте как минимум один раз. Например, формула xy+z вернет None строке xy, и вернет положительный результат строкам xyz, xyyyyz.

Знак ? - означает, что данный символ может встретиться либо 0 либо 1 раз. не более. Например, xy?z будет соответствовать только строкам xz, и xyz.

Теперь, если скомбинировать эти знания с коллекцией символов, мы можем сделать следующее:

```
text1 = "x_FK_0_z"
text2 = "xabba_FK_5_z"
text3 = "xb_FK__z"
formula = re.compile("x[a-z]*_FK_[0-9]?_z")
print ( formula.match(text3) ) # Все 3 текста совпадут с нашей формулой
```

Есть еще больше специальных символов, которые позволяют найти определенное количество повторений символа. Например **xy{2,5}z** найдет только те слова, у которых у встречается от 2 до 5 раз подряд. Т.е. **xyyyz** сработает, а вот **xyyyyyy** уже не сработает, т.к. у встречается 6 раз. Чтобы указать точное число повторений- просто используем одну цифру в фигурных скобках, например **y{3}** значит что символ у должен повториться 3 раза.

Выборка из слов (OR)

Если мы хотим найти слова, у которых в середине присутствует слово car либо bus, можем воспользоваться формулой (car|bus).

```
text1 = "x_car_z"
text2 = "x_bus_z"
formula = re.compile("x_(car|bus)_z")
print ( formula.match(text2) )
```

Начало и конец строки

Иногда мы хотим быть уверены в том, что наша формула покрывает абсолютно всю строку от начала до конца. Мы можем использовать специальные символы, ^ - обозначает начало строки, и $ - обозначает конец строки.

```
text1 = "abba"  # Correct !
text2 = "zabba" # Wrong !
formula = re.compile("^abba$")
print ( formula.match(text1) )
```

Группы

Группы - это очень полезное свойство регулярных выражений, позволяющее вернуть не просто результат совпадения строки, а определенные части строки, которые мы потом используем для иных целей. Группы пишутся в скобках. Порядок нумерации групп определяется порядком их появления в формуле. Под номером 0 идет весь текст, который мы сравнили с формулой. Под номером 1 уже идет первая группа формулы, и т.д. Но проще посмотреть пример.

Представим, что у нас есть путь к файлу, и нам нужно получить из этого пути определенную информацию. Например:

```
/net/homedir/rvolodin/dev/project/sequences/LS_ABB_P/shots/LS_ABB_0021_001
```

Нам необходимо получить из части **/LS_ABB_P/** название сиквенции **ABB**, а также название шота и его номер **ABB_0021**. Причем названия сиквенций и шотов могут отличаться.

```
text1 = "/net/homedir/rvolodin/dev/project/sequences/LS_ABB_P/shots/LS_ABB_0021_001"
text2 = "/net/homedir/rvolodin/dev/project/sequences/LX_CCG_P/shots/LX_CCG_0022_001"
text3 = "/net/homedir/rvolodin/dev/project/sequences/LZ_ABC_P/shots/LZ_ABC_0023_001"

formula = re.compile(".*/L[A-Z]_([A-Z]{3})_P.*L[A-Z]_([A-Z]{3}_[0-9]{4})_[0-9]{3}")
result = formula.match(text1)

print (result.group(1)) # >> ABB
print (result.group(2)) # >> ABB_0021
```

Формула кажется очень сложной, но давайте разберем ее подробнее:

- **.*** означает любое количество любых символов
- */* - это уже символ текста который встречается в тексте несколько раз
- **L[A-Z]_([A-Z]{3})_P** означает набор символов, под который подходят строки *LS_ABB_P*, *LX_CCG_P* и *LZ_ABC_P*. Скобки обозначают группу, в которой хранится та под-строка, которую мы хотим получить.
- **L[A-Z]_([A-Z]{3}_[0-9]{4})_[0-9]{3}** соответствует как *LS_ABB_0021_001*, так и *LX_CCG_0022_001* и *LZ_ABC_0023_001*.

Бонус: одно из заданий Double Negative, которое они мне дали на интервью

Задача: разбить текст по специальным символам **!.?** в список и вернуть число, представляющее из себя максимальное количество слов среди всех элементов этого списка.

```
import re

# Split string with symbols !.? and get the substring with maximum of words

def solution(S):
    result = re.split('[!.?]+', S)

    #split string by any of [! . ?] symbols
    # do it when they appear like . or ... or ???

    out = 0
    substring = None

    for i in result:

        # first check if this split i is not spaces only
        sub = i.strip() #delete spaces from both sides of the string

        if not len(sub): #if there were only spaces - continue
            continue

        sub = ' '.join(sub.split()) # join "sub" elements with ' '. Becomes a single word

        length = len(sub.split(" ")) # get number of words in i

        if length > out:
            # if it gives better result
            out = length
            substring = sub

    return out


print (solution("Forget  CVs..Save time . x x"))
```