Лекція 33

Регулярні вирази

Що таке регулярні вирази ?

Просто кажучи, регулярний вираз - це послідовність символів, яка використовується для пошуку і заміни тексту в рядку або файлі.

Їх підтримує безліч мов загального призначення: Python, Perl, Java і т.і. Тому вивчення регулярних виразів рано чи пізно стане в нагоді.

У мові Python використовувати регулярні вирази дозволяє модуль **re**.

Перед використанням функцій із цього модуля необхідно підключити модуль за допомогою інструкції:

```
import re
```

Найчастіше регулярні вирази використовуються для:

пошуку в рядку;
розбиття рядка на підрядки;
заміни частини рядка.

Основні функції модуля re

- `re.match()` – ця функція шукає відповідність заданому шаблону на початку рядка.
- `re.search()` – шукає відповідність заданому шаблону у всьому рядку, але повертає тільки першу знайдену відповідність.
- `re.findall()` – повертає список всіх знайдених відповідностей.
- `re.split()` – розділяє рядок на підрядки за заданим шаблоном
- `re.sub()` – шукає шаблон у рядку та замінює його на заданий підрядок
- `re.compile()` – дозволяє зібрати регулярний вираз в окремий об'єкт, який може бути використаний для пошуку. Це також дозволяє позбутись переписування одного і того ж виразу

Результат пошуку функціями re.match та re.search

Якщо пошук не досяг успіху, то функції повертають None.

Якщо шуканий підрядок знайдено,то функції повертають об’єкт типу <class '_sre.SRE_Match'>.

Цей об’єкт має наступні методи:

Метод **group ()** для виводу знайденого підрядка

Метод **start ()** для виводу початкової позиції знайденого рядка.

Метод **end()** для виводу кінцевої позиції знайденого рядка.

Використовуємо «r» перед рядком шаблону, щоб показати, що це «сирий» рядок в Python.

Приклад 1. Застосування функції `re.match()`

```
import re
result = re.match(r'Мій', 'Мій перший пошук')
result1 = re.match(r'перший', 'Мій перший пошук')
if result:
    print(type(result))
    print ("Знайдено:",result.group(0))
    print ("Початковий індекс:",result.start())
    print ("Кінцевий індекс:",result.end())
else: print ("Результат:",result)
if result1: print (result1.group(0))
else: print ("Результат:",result1)
```

Результати роботи

```
<class '_sre.SRE_Match'>
Знайдено: Мій
Початковий індекс: 0
Кінцевий індекс: 3
Результат: None
```

Приклад 2. Застосування функції `re.search`

```
import re
result = re.search(r'другий', 'Мій другий пошук')
result1 = re.search(r'перший', 'Мій другий пошук')
if result:
    print(type(result))
    print ("Знайдено:",result.group(0))
    print ("Початковий індекс:",result.start())
    print ("Кінцевий індекс:",result.end())
else: print ("Результат:",result)
if result1: print (result1.group(0))
else: print ("Результат:",result1)
```

Результати роботи

```
<class '_sre.SRE_Match'>
Знайдено: другий
Початковий індекс: 4
Кінцевий індекс: 10
Результат: None
```

Результати пошуку за функцією `re.findall`

Якщо пошук не досяг успіху, то функція повертає пустий список.

Якщо шукані підрядки знайдено, то функція повертає об’єкт типу <class 'list'>.

Приклад 3. Застосування функції `re.findall`

```
import re
result = re.findall(r'ш', 'Я шукаю перший раз')
result1 = re.findall(r'перший', 'Мій другий пошук')
if result:
    print(type(result))
    print ("Знайдено:",result)
else: print ("Результат:",result)
if result1: print (result1)
else: print ("Результат:",result1)
```

Результат: <class 'list'>
Знайдено: ['ш', 'ш']
Результат: []

Що таке шаблон?

Ми розглядали шаблон як послідовність символів.
В Python можливо задати також шаблон на простій мові, яка дозволяє шукати підрядок у рядку за певними правилами.

Використовуючи цю мову, вказуємо правила для довільної кількості можливих рядків, які ми хочемо перевірити.

Ці рядки можуть містити англійські фрази, адреси електронної пошти, команди, все що завгодно.

За допомогою шаблона можна одержати відповідь на такі питання:

«Чи відповідає цей рядок шаблону?»,

«Чи збігається шаблон з цим рядком частково?».

Метасимволи

Метасимволи – це спеціальні символи, які вказують на те, що задана деяка певна умова. Вони можуть впливати на інші частини регулярного виразу, змінюючи їх значення.

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

`[` та `]` – використовуються для визначення класу символів, що є набором символів, з якими шукаємо збіг.

Символи можуть бути перераховані окремо, або у вигляді деякого діапазону символів, позначеного першим і останнім символом, розділених знаком '-'.

Наприклад:

[abc]- буде відповідати будь-якому з символів a, b або c;

[a-c] -аналогічно можна задати діапазон.

[a-z] – всі малі латинські літери.

Приклади класів в [ ]

[09] – відповідає числу 0 або 9;

[0-9] – відповідає будь-якому числу від 0 до 9;

[абв] – відповідає буквам «а», «б» і «в»;

[а-яґ] – відповідає будь-якій букві від «а» до «я»;

[АБВ] – відповідає буквам «А», «Б» і «В»;

[А-ЯҐ] – відповідає будь-якій букві від «А» до «Я»;

[а-яА-ЯґҐ] – відповідає будь-якій кириличній букві ;

ПРИМІТКА

Буква «ґ» не входить у діапазон [а–я], а буква «**Ґ**» – у діапазон [А-Я].

Значення в дужках інвертується, якщо після першої дужки вставити символ ^. У такий спосіб можна вказати символи, яких не повинне бути на цьому місці в рядку:

[^09]- не цифра 0 або 9;

[^0-9]- не цифра от 0 до 9;

[^а-яА-ЯґҐa-zA-Z]- не буква.

Поведінка метасимволів в [ ]

Метасимволи не активні всередині класів.

Наприклад.

[akm $] - буде відповідати будь-якому з символів 'a', 'k', 'm' або '$'.

Знак '$' це зазвичай метасимвол (як видно зі списку символів вище), але всередині класу символів він позбавляється своєї особливої природи.

Для того, щоб знаходити відповідність символам поза класом, на початку класу додається символ '^'. Наприклад, вираз [^5] відповідає будь-якому символу, крім '5'.

Метасимвол '\'

1.Метасимвол \ використовується для екранування метасимволов, щоб їх можна було використовувати в шаблонах.

Наприклад.

Нехай потрібно знайти відповідність **[** або ****.

Для того щоб позбавити їх своєї особливої ролі метасимволов, перед ним треба поставити зворотний слеш: **\[** або ****.

2.Деякі зі спеціальних послідовностей, що починаються з '\' являють зумовленими наборами символів, які корисні для скорочення регулярного виразу

Наступні зумовлені послідовності є їх підмножиною.

Зумовлені послідовності з метасимволом \

\d - відповідає будь-якій цифрі; еквівалент класу [0-9].

\D - відповідає будь-якому нечисловому символу; еквівалент класу [^0-9].

\s- відповідає будь-якому символу whitespace; еквівалент [\t\n\r\f\v].

\S - відповідає будь-якому не-whitespace символу; еквівалент [^\t\n\r\f\v].

\w - відповідає будь-якій букві або цифрі; еквівалент [a-zA-Z0-9_].

\W - навпаки; еквівалент [^a-zA-Z0-9_].

\b – прив'язка до початку слова (початком слова вважається пробіл або будь-який символ, що не є буквою, цифрою або знаком підкреслення);

\B– прив'язка до позиції, що не є початком слова.

Ці послідовності можуть бути включені в клас символів. Наприклад

`[\s,.]` - клас, який буде відповідати будь-якому `whitespace`-символу або комі (,) або точці (.).

Метасимвол '.'

Метасимвол '.' відповідає всім символам, крім символу нового рядка, але є альтернативний режим (re.DOTALL), при якому буде включатися і '.' .

Метасимвол '.' використовується там, де необхідно порівняти будь-які символи.

Метасимвол прив’язки

^ – прив'язка до початку рядка або підрядка.

$ – прив'язка до кінця рядка або підрядка.

\A – прив'язка до початку рядка

\Z – прив'язка до кінця рядка

Метасимвол альтернативи |

Метасимвол | дозволяє зробити вибір між альтернативними значеннями.

Выраз **n|m** відповідає одному із символів: **n** або **m**.

Метасимволи входжень

{n} – n входжень символу в рядок.

Наприклад.

Шаблон `r"[0-9]{2}$"` відповідає двом входженням будь-якої цифри;

{n,m} – не менше n і не більше m входжень символу в рядок. Числа вказуються через кому без пробілу. Наприклад, шаблон `r"[0-9]{2,4}$"` відповідає від двох до чотирьох входжень будь-якої цифри;

Метасимволи входжень (продовження)

***** – нуль або більше число входжень символу в рядок. Еквівалентно комбінації `{0,};`

+ – одне або більше число входжень символу в рядок. Еквівалентно комбінації `{1,};`

? – жодного або одне входження символу в рядок. Еквівалентно комбінації `{0,1}`.

Синтаксис регулярних виразів

Створити відкомпільований шаблон регулярного виразу дозволяє функція `compile()`.

Функція має наступний формат:

`<Об'єкт пошуку>=`**`re.compile(<Регулярний вираз>[,<Модифікатор>])`**

У параметрі `<Модифікатор>` можуть бути зазначені прапори компіляції.

Прапори компіляції дозволяють змінювати деякі аспекти того, як працюють регулярні вирази.

Прапори доступні в модулі під двома іменами: довгим, таким як IGNORECASE і коротким, в однобуквені формі, таким як I. Кілька прапорів можуть бути задані у формі двійкового АБО; наприклад re.I | re.M встановлює прапори I і M.

Прапори компіляції

ASCII, A вибирає тільки ASCII-символи
DOTALL, S відповідність така ж як '.', Тобто з будь-яким символом, але при включенні цього прапора, в розгляд додається і символ нового рядка.
IGNORECASE, I відповідність без урахування регістру; Наприклад, [A-Z] буде також відповідати і рядковим буквах, так що Spam буде відповідати Spam, spam, spAM і так далі.

LOCALE, L - робить \ w, \ W, \ b, \ B залежними від локалізації. Наприклад, якщо ви працюєте з текстом українською, і хочете написати \w + для того, щоб знаходити слова, але \w шукає тільки символи з множини [А-Яа-я] і не буде шукати 'ґ". Якщо система налаштована правильно і задано прапор re.L , 'ґ" також буде розглядатися як буква.

Прапори компіляції (продовження)

MULTILINE, M – зазвичай ^ шукає відповідність тільки на початку рядка, а $ тільки в кінці безпосередньо перед символом нового рядка (якщо такі є). Якщо цей прапор вказано, ^ порівняння відбувається у всіх рядках, тобто і на початку, і відразу ж після кожного символу нового рядка. Аналогічно для $.

UNICODE, U - робить \ w, \ W, \ b, \ B, \ d, \ D, \ s, \ S відповідними таблиці Unicode.

VERBOSE, X включає багатослівні (докладні) регулярні вирази, які можуть бути організовані більш ясно і зрозуміло. Якщо вказаний цей прапор, прогалини в рядку регулярного виразу ігнорується, крім випадків, коли вони є в класі символів або їм передує неекранований зворотний слеш; дозволяє поміщати в регулярні вирази коментарі, що починаються з '#', які будуть ігноруватися движком.

Основні методи об’єкта пошуку

<**Об’єкт пошуку**>.match() – цей метод шукає відповідність заданому шаблону на початку рядка.

<**Об’єкт пошуку**>.search() – шукає відповідність заданому шаблону у всьому рядку, але повертає тільки першу знайдену відповідність.

<**Об’єкт пошуку**>.findall() – повертає список всіх знайдених відповідностей.

<**Об’єкт пошуку**>.fullmatch()– виконує перевірку, чи відповідає переданий рядок регулярному виразу цілком. Підтримка цього методу з'явилася в Python 3.4.

<**Об’єкт пошуку**>.finditer()–аналогічно findall(), але повертає ітератор, а не список

Приклад 4. Пошук єдиної відповідності і пошук одної (або більше) відповідності для кожної букви

```
import re
p=re.compile(r"[а-яґ]", re.I | re.U)
p1=re.compile(r"[а-яґ]+", re.I | re.U)
s = p.search("АБВГДЕ")
s1 = p1.search("АБВГДЕ")
s2 = p1.search("абвгдАБВГДЕ")
if s:print(s.span())
    print(s.group(0))
if s1: print(s1.span())
    print(s1.group(0))
if s2: print(s2.span())
    print(s2.group(0))
```

Результат:

(0, 1)
А
(0, 6)
АБВГДЕ
(0, 11)
абвгдАБВГДЕ

Приклад 5. Перевірка на розпізнавання символу переходу на новий рядок з різними прапорами

```
mport re
def check(a):
    if a:print(repr(a.group()))
        print(a.span())
    else: print(a)
p = re.compile(r".$")
check(p.search("\n"))
p1 = re.compile(r".$", re.M)
check(p1.search("\n"))
p2 = re.compile(r".$", re.S)
check(p2.search("\n"))
```

Результат:

```
None
None
'\n'
(0, 1)
```

Прив'язку до початку й кінця рядка слід використовувати, якщо рядок повинен повністю вдповідати регулярному виразу. Наприклад, для перевірки, чи містить рядок число.

Приклад 6. В рядку мають бути тільки цифри

```
import re
p = re.compile (r"^[0-9]+$") #одне або більше входження
print("Знайдено" if p.search("1234567890") else "Hi")
print("Знайдено" if p.search("q1234567890") else "Hi")
print("Знайдено" if p.search("1234567890q") else "Hi")
print("Знайдено" if p.search("1234567q890") else "Hi")
print("Знайдено" if p.search("") else "Hi")
```

Результат:

```
Знайдено
Hi  Hi  Hi  Hi
```

Якщо забрати прив'язку до початку й кінця рядка, то будь-який рядок, що містить хоча б одну цифру, буде розпізнаний як Число

Приклад 7. Пошук без прив'язки до початку й кінця рядка

```
import re # Підключаємо модуль
p = re.compile(r"[0-9]+", re.S)
if p.search("Строка245"):
    print("Число в рядку")
else:
    print("Немає числа в рядку")
```

Результат:

```
Число в рядку
```

Крім того, можна вказати прив'язку тільки до кінця рядка

Приклад 8. Пошук з прив’язкою тільки до кінця рядка

```
import re # Підключаємо модуль
p = re.compile(r"[0-9]+$", re.S)
if p.search("Рядок245"):
    print("Є число наприкінці рядка")
else:
    print("Немає числа наприкінці рядка")
```

Результат: Є число наприкінці рядка

Приклад 9. Пошук з прив’язкою тільки до початку рядка

```
import re # Підключаємо модуль
p = re.compile(r"^[0-9]+", re.S)
if p.search("Рядок245"):
    print("Є число на початку рядка")
 else:
    print("Немає числа на початку рядка")
```

Результат: Немає числа на початку рядка

Приклад 10. Прив’язка до початку рядка за допомогою \b

```
import re
p = re.compile(r"\bpython")
print("Знайдено" if p.search("python") else "Hi")
print("Знайдено" if p.search("pythonware") else "Hi")
print("Знайдено" if p.search("thon") else "Hi")
print("Знайдено" if p.search("pyth") else "Hi")
```

Результат: Знайдено Знайдено Hi Hi

```
#Прив’язка до початку  і кінця рядка за допомогою \b
import re
p = re.compile(r"\bpython\b")
print("Знайдено" if p.search("python") else "Hi")
print("Знайдено" if p.search("pythonware") else "Hi")
print("Знайдено" if p.search("thon") else "Hi")
print("Знайдено" if p.search("pyth") else "Hi")
```

Результат: Знайдено Hi Hi Hi

Приклад 11. В рядку мають бути тільки цифри або пустий рядок

```
import re
p = re.compile (r"^[0-9]*$")#нуль або більше входжень
print("Знайдено" if p.search("") else "Hi")
print("Знайдено" if p.search("12334567890") else "Hi")
print("Знайдено" if p.search("w12334567890") else "Hi")
print("Знайдено" if p.search("123w3457890") else "Hi")
print("Знайдено" if p.search("wwwww") else "Hi")
```

Результат:

Знайдено

Знайдено

Hi

Hi

Hi

Приклад 12. В рядку мають бути тільки цифри або пустий рядок

```
import re
#жодного або одне входження
p = re.compile (r"^[0-9]?$") print("Знайдено"
if p.search("") else "Ні")
print("Знайдено" if p.search("1") else "Ні")
print("Знайдено" if p.search("6") else "Ні")
print("Знайдено" if p.search("w") else "Ні")
print("Знайдено" if p.search("w1") else
"Ні")
```

Результат:

Знайдено

Знайдено

Знайдено

Ні

Ні

Приклад 13. Перевірка формату дати

```
import re # Підключаємо модуль
d = "29,12.2016"  # Замість крапки вказана кома
p = re.compile(r"^[0-3][0-9].[01][0-9].[12][09][0-9][0-9]$")
if p.search(d): print("Дата введена правильно")
else: print("Дата введена неправильно")
p = re.compile(r"^[0-3][0-9]\.[01][0-9]\.[12][09][0-9][0-9]$")
if p.search(d): print("Дата введена правильно")
else:   print("Дата введена неправильно")
p = re.compile (r"^[0-3][0-9][.][01][0-9][.][12][09][0-9][0-9]$")
if p. search(d):   print("Дата введена правильно")
else:  print("Дата введена неправильно")
```

Результат:

```
Дата введена правильно {крапка-метасимвол}
Дата введена неправильно
Дата введена неправильно
```

Приклад 14. Вплив модифікаторів (прапорів)

```
import re
# Крапка не відповідає \n
p = re.compile(r"^.+$")
print(p.findall("str1\nstr2\nstr3"))

# Тепер крапка відповідає \n
p = re.compile(r"^.+$",re.S)
print(p.findall("str1\nstr2\nstr3"))

# Багаторядковий режим
p = re.compile(r"^.+$",re.M)
print(p.findall("str1\nstr2\nstr3"))
```

Результат

```
[]
['str1\nstr2\nstr3']
['str1', 'str2', 'str3']
```

Метасимвол | дозволяє зробити вибір між альтернативними значеннями. Вираз `n|m` відповідає одному із символів: `n` або `m`.

Приклад 15.

```
import re
p = re.compile(r"червон((а)|(е))")
print("Знайдено" if p.search("червона") else "Ні")
print("Знайдено" if p.search("червоне") else "Ні")
print("Знайдено" if p.search("червоний") else "Ні")
```

Результат:

```
Знайдено
Знайдено
Ні
```

Пошук findall() по шаблону

Усі метасимволи є «жадібними». При пошуку відповідності шукається найдовший підрядок, відповідно до шаблону, і не враховуються більш короткі відповідності. Розглянемо це на прикладі й одержимо вміст усіх тегів , разом з тегами:

Приклад 16.

```
import re
#* - 0 або більше
s = "<b>Text1</b> Text2\n <b>Text3</b>"
p = re.compile(r"<b>.*</b>", re.S)
print(p.findall(s))
```

Результат:

```
['<b>Text1</b> Text2\n <b>Text3</b>']
```

Даний рядок починається на і закінчується на

Обмеження пошуку findall() по шаблону

Замість бажаного результату ми одержали повністю рядок. Щоб обмежити «жадібність», необхідно після квантифікатора вказати символ ?

Приклад 17. Обмежена «жадібність» модифікатора

```
import re
#* - 0 або більше
#? - жодного або одне
s = "<b>Text1\n</b>Text2<b>Text3</b>"
p = re.compile(r"<b>.*?</b>", re.S)
print(p.findall(s))
```

Результат:

```
['<b>Text1\n</b>', '<b>Text3</b>']
```

Цей код вивів те, що ми шукали.

Одержати текст без тегів

Якщо необхідно одержати вміст без тегів, то потрібний фрагмент всередині шаблону слід розмістити всередині круглих дужок

Приклад 18.

```
import re
#* - 0 або більше
#? - жодного або одне
s = "<b>Text1\n</b>Text2<b>Text3</b>"
p = re.compile(r"<b>(.*?)</b>", re.S)
print(p.findall(s))
```

Результат: `['Text1\n', 'Text3']`

Угрупування фрагментів усередині шаблону

Для угруповання фрагментів усередині шаблону будемо користуватися круглими дужками

У цьому випадку не потрібно, щоб фрагмент запам'ятовувався й був доступний у результатах пошуку але щоб уникнути захоплення фрагмента

Для уникнення захоплення слід після відкриваючої круглої дужки розмістити символи ?:

Приклад.

```
s = "Учні та  учителі в школі"
```

Шаблон пошуку: `([а-я]+((ителі)|(ні)))`

Зовнішня дужка захоплює елементи «**Учні**», «**учителі**»

Внутрішні дужки захоплюють елементи «**ителі**», «**ні**»

Потрібно виключити захоплення цих фрагментів

Приклад 19. Обмеження захоплення фрагмента

```
import re
#* - 0 або більше
#? - жодного або одне
s = "Учні та  учителі в школі"
p = re.compile(r"([а-я]+((ителі)|(ні)))", re.I)
print(p.findall(s))
p1 = re.compile(r"([а-я]+(?:(?:ителі)|(?:ні)))", re.I)
print(p1.findall(s))
```

Результат:

```
[('Учні', 'ні', '', 'ні'), ('учителі',
'ителі', 'ителі', '')]
['Учні', 'учителі']
```

Пояснення до прикладу 19

У першому прикладі ми одержали список із двома елементами. Кожний елемент списку є кортежем, що містить чотири елементи.

Усі ці елементи відповідають фрагментам, укладеним у шаблоні в круглі дужки. Перший елемент кортежу містить фрагмент, розташований у перших круглих дужках, другий – у других круглих дужках і т. д. Три останні елементи кортежу є зайвими.

Щоб вони не виводилися в результатах, ми додали символи `?:` після кожної відкриваючої круглої дужки. У результаті список складається тільки із фрагментів, що повністю відповідають регулярному виразу.