Лекція 34

Регулярні вирази(продовження)

Зворотне посилання на фрагмент

До знайденого фрагмента в круглих дужках усередині шаблону можна звернутися за допомогою механізму зворотних посилань. Для цього порядковий номер круглих дужок у шаблоні вказують після слеша – наприклад, так: `\1`. Нумерація дужок усередині шаблону починається з `1`. Для прикладу одержимо текст між однаковими парними тегами.

Приклад 1.

```
import re
s = "<b>Один</b> Два <I>Три</I> <p>Чотири</p>"
p = re.compile(r"<([a-z]+)>(.*?)</\1>", re.S | re.I)
print(p.findall(s))
```

Результат:

```
[('b', 'Один'), ('I', 'Три'), ('p', 'Чотири')]
```

Використання іменованих фрагментів

Фрагментам усередині круглих дужок можна дати імена. Для цього після відкриваючої круглої дужки слід указати комбінацію символів `?P<name>`. Як приклад розберемо e-mail на складові частини:

Приклад 2. Іменовані фрагменти

```
import re
email = "test@ukr.net"
p = re.compile(r"(?P<name>[a-z0-9]+)@(?P<host>[a-z0-
9]+\.[a-z]{2,6})", re.I )
z=p.search(email)
print(z. group ("name"))
print(z.group("host"))
```

Результат:

```
test
ukr.net
```

Звернення до іменованих фрагментів усередині шаблону

Щоб всередині шаблону звернутися до іменованих фрагментів, використовується наступний синтаксис: (?P=name). Для прикладу одержимо текст між однаковими парними тегами

Приклад 3. Доступ до іменованих фрагментів усередині шаблону:

```
import re
s = "<b>Text1</b>Text2<I>Text3</I>"

p = re.compile(r"<(?P<tag>[a-z]+)>(.*?)</(?P=tag)>", re.S | re.I)

print(p.findall(s))
```

Результат:

```
[('b', 'Text1'), ('I', 'Text3')]
```

Можливі конструкції усередині круглих дужок

`(?ailmsux)` – дозволяє встановити опції регулярного виразу.
Букви `a, i, l, m, s, u` і `x` мають таке ж призначення, що й однойменні модифікатори у функції `compile ();`

`(?#...)`– коментар. Текст всередині круглих дужок ігнорується;
`(?=...)`– позитивний перегляд вперед.
`(?!...)` – негативний перегляд уперед.
`(?<=...)` – позитивний перегляд назад.
`(?<!...)` – негативний перегляд назад.

Виведемо всі слова, після яких розташована кома:

Приклад 4. позитивний перегляд вперед

```
import re
s = "Перша_кома, Друга_кома, Без_коми"
p = re.compile(r"\w+(?=[,])", re.S | re.I)
print(p.findall(s))
```

Результат:

```
['Перша_кома', 'Друга_кома']
```

Виведемо всі слова, після яких немає коми:

Приклад 5. негативний перегляд уперед

```
import re
s = "text1, text2, text3 text4"
p = re.compile(r"[a-z]+[0-9](?![,])",re.S | re.I)
print(p.findall(s))
```

Результат:

```
['text3', 'text4']
```

Виведемо всі слова, перед якими розташована кома з пробілом:

Приклад 6. позитивний перегляд назад

```
import re
s = "text1, text2, text3 text4"
p = re.compile(r"(?<=[,][ ])[a-z]+[0-9]", re.S | re.I)
print(p.findall(s))
```

Результат:

```
['text2', 'text3']
```

Виведемо всі слова, перед якими розташований пробіл, але перед пробілом немає коми:

Приклад 7. негативний перегляд назад

```
import re
s = "text1, text2, text3 text4"
p = re.compile(r"(?<![,]) ([ ][a-z]+[0-9])", re.S | re.I)
print(p.findall(s))
```

Результат: `[' text4']`

Приклад послідовності з дефісами

Необхідно одержати всі слова, розташовані після дефіса,

причому перед дефісом і після слів повинні слідувати «пропускові» символи:

Приклад 8.

```
import re
s = "-word1 -word2 -word3 -word4 -word5"
p=re.compile(r"\s\-([a-z0-9]+)\s",re.S | re.I)
print(p.findall(s))
```

Результат:

```
['word2', 'word4']
```

Як видно із прикладу, ми одержали тільки два слова замість п'яти. Перше й останнє слова не потрапили в результат, тому що розташовані на початку й наприкінці рядка.

Щоб ці слова потрапили в результат, необхідно додати альтернативний вибір `(^ | \s)` – для початку рядка

і `( \s | $)` – для кінця рядка. Щоб знайдені вирази всередині круглих дужок не потрапили в результат, слід додати символи `?:` після відкриваючої дужки:

Приклад 9.

```
import re
s = "-word1 -word2 -word3 -word4 -word5"
p = re.compile(r"(?:^|\s)\-([a-z0-9]+)(?:\s|$)", re.S | re.I)
print(p.findall(s))
```

Результат:

```
['word1', 'word3', 'word5']
```

Перше й останнє слова успішно потрапили в результат. Чому ж слова word2 і word4 не потрапили в список збігів – адже перед дефісом є пробіл і після слова є пробіл?

Щоб зрозуміти причину, розглянемо пошук по кроках. Перше слово успішно потрапляє в результат, тому що перед дефісом розташований початок рядка, і після слова є пробіл. Після пошуку покажчик переміщається, і рядок для подальшого пошуку набуде наступного вигляду:

"-word1 <Покажчик>-word2 –word3 -word4 -word5"

Зверніть увагу на те, що перед фрагментом -word2 більше немає пробілу, і дефіс не розташований на початку рядка. Тому наступним збігом буде слово word3, і покажчик знову буде переміщений:

"-word1 -word2 –word3 <Покажчик>-word4 -word5"

Знову перед фрагментом `-word4` немає пробілу, і дефіс не розташований на початку рядка. Тому наступним збігом буде слово `word5`, і пошук буде завершений. Таким чином, слова `word2` і `word4` не попадають у результат, оскільки пробіл до фрагмента вже був використаний у попередньому пошуку.

Щоб цього уникнути, слід скористатися позитивним переглядом уперед `(?=...)`:

Приклад 10.

```
import re
s = "-word1 -word2 -word3 -word4 -word5"
p = re.compile(r"(?:^|\s)\-([a-z0-9]+)(?=\s|$)", re.S | re.I)
print(p.findall(s))
```

Результат:

['word1', 'word2', 'word3', 'word4', 'word5']

У цьому прикладі ми замінили фрагмент `(?: \s | $)` на `( ?=\s | $)`.

Тому всі слова успішно потрапили в список збігів.

Огляд методів та функцій пошуку

1. Формат функції re.match() модуля re:

re.match (<Шаблон>, <Рядок>[, <Модифікатор>])

У параметрі <Шаблон> вказується рядок з регулярним виразом або скомпільований регулярний вираз.

У параметрі рядок<Рядок> задаємо рядок, у якому відбувається пошук.

У параметрі <Модифікатор> можна вказати прапори, використовувані у функції compile().

Якщо відповідність знайдена, то повертається <**об'єкт пошуку**>, а якщо ні, то повертається значення None.

Приклад 11.

```
import re
def check(a):
    if a: print("Знайдено")
    else: print("Ні")

p1 = r"[0-9]+"
check(re.match(p1, "str123"))
check(re .match(p1, "123str"))
p2 = re.compile(r"[0-9]+")
check(re.match(p2, "123str"))
```

Результат роботи:

```
Ні
Знайдено
Знайдено
```

2. Формат методу <об’єкт пошуку>.match модуля re:

<об’єкт пошуку>.**match**(<Рядок>[,<Початкова позиція>[,<Кінцева позиція>]])

Якщо відповідність знайдена, то повертається об'єкт Match, а якщо ні, то повертається значення None.

Приклад 12.

```
import re
def check(a):
    if a: print("Знайдено")
    else: print("Ні")
tamp = re.compile(r"[0-9]+")
check(tamp.match("str123"))
check(tamp.match("str123",3))
check(tamp.match("123str"))
```

Результат роботи:

```
Ні
Знайдено
Знайдено
```

3. Формат функції re.search() модуля re:

`re.search``(<Шаблон>, <Рядок>[,<Модифікатор>))`

У параметрі <`Шаблон`> вказується рядок з регулярним виразом або скомпільований регулярний вираз.

У параметрі <`Модифікатор`> можна вказати прапори для використання у функції `compile()` .

У параметрі рядок<Рядок> задаємо рядок, у якому відбувається пошук.

Якщо відповідність знайдена, то повертається об'єкт `Match`, а якщо ні, то повертається значення `None`.

Приклад 13.

```
import re
def check(a):
    if a: print("Знайдено")
    else: print("Hi")
p1 = r"[0-9]+"
check(re.search(p1, "str123"))
p2 = re.compile(r"[0-9]+")
check(re.search(p2, "str123"))
check(re.search(p2, "123str"))
```

Результат роботи:

```
Знайдено
Знайдено
Знайдено
```

4. Формат методу <об'єкт пошуку>.**search** модуля re:

<об'єкт пошуку>.search (<Рядок>[,<Початкова позиція>[,<Кінцева позиція>]])

search() – перевіряє відповідність із будь-якою частиною рядка. Якщо знайденj, то повертається об'єкт Match, а якщо ні, то повертається значення None.

Приклад 14.

```
import re
def check(a):
    if a: print("Знайдено", end=' ')
    else: print("Ні", end=' ')
tamp = re.compile(r"[0-9]+")
check(tamp.search("str123"))
check(tamp.search("123str",3))
check(tamp.search("123str"))
```

Результат роботи:

Знайдено Ні Знайдено

5. Формат функції re. **`Fullmatch()`** модуля re:

`re.fullmatch``(<Шаблон>,<Рядок>[,<Модифікатор>])`

У параметрі <Шаблон> вказується рядок з регулярним виразом або скомпільований регулярний вираз.

У параметрі <Модифікатор> можна вказати прапори, використовувані у функції `compile()` .

У параметрі рядок<Рядок> задаємо рядок, у якому відбувається пошук.

Якщо рядок повністю збігається з шаблоном, повертається об'єкт `Match`, а якщо ні, то повертається значення `None`.

Приклад 15.

```
import re
def check(a):
    if a: print("Знайдено")
    else: print("Ні")
p = "[Pp]ython"
check(re.fullmatch (p,"Python"))
check(re.fullmatch (p,"python"))
check(re.search(p, "py"))
check(re.search(p, "thon"))
```

Результат роботи:

```
Знайдено
Знайдено
Ні
Ні
```

6. Формат методу <об'єкт пошуку>.fullmach модуля re:

<об'єкт пошуку>.fullmach(<Рядок>[,<Початкова позиція>[,<Кінцева позиція>]])

fullmatch()– виконує перевірку, чи відповідає переданий рядок регулярному виразу цілком.

Приклад 16.

```
import re
def check(a):
    if a: print("Знайдено", end=' ')
    else: print("Ні", end=' ')
tamp = re.compile("[Pp]ython")
check(tamp.fullmatch("Python"))
check(tamp.fullmatch("python"))
check(tamp.fullmatch("py"))
check(tamp.fullmatch("nothpy"))
check(tamp.fullmatch("Pythonware",0,6))
```

Результат роботи: Знайдено Знайдено Ні Ні Знайдено

Приклад застосування регулярних виразів

Приклад 17. Додавання невизначеної кількості чисел

```
import re
print("Введіть слово 'stop' для отримання результату")
suma = 0
p = re.compile(r"^[-]?[0-9]+$", re.S)
while True:
    x = input("Введіть число: ")
    if x == "stop":
        break  # Вихід з циклу
    if not p.search(x):
        print("Необхідно ввести число, а не рядок!")
        continue # Переходимо на наступну ітерацію циклу
    x = int(x) # Перетворюємо рядок на число
    suma += x
print("Сума чисел дорівнює:", suma)
```

Атрибути та методи об’єкта `Match`

Об'єкт `Match`, що повертається методами (функціями) `match()` і `search()`, має наступні властивості й методи:

`<об’єкт пошуку>.re.groups` - кількість груп у шаблоні;

`<об’єкт пошуку>.re.groupindex` - словник з назвами груп і їх номерами;

`<об’єкт пошуку>.re.pattern` - початковий рядок з регулярним виразом;

`<об’єкт пошуку>.re.flags` - комбінація прапорів, заданих при створенні регулярного виразу у функции `compile()`, і прапорів, зазначених у самому регулярному виразі, у конструкції `(?ailmsux)`;

<об'єкт пошуку>.**string** – значення параметра <Рядок> у методах (функціях) `match()` і `search ();`

<об'єкт пошуку>.**pos** – значення параметра <Початкова позиція> у методах `match()` і `search()` ;

<об'єкт пошуку>.**endpos** – значення параметра <Кінцева позиція> у методах `match()` і `search()` ;

<об'єкт пошуку>.**lastindex** – повертає номер останньої групи або значення `None`, якщо пошук завершився невдачею;

<об'єкт пошуку>.**lastgroup** – повертає назва останньої групи або значення `None`, якщо ця група не має ім'я, або пошук завершився невдачею.

Приклад 18. `import re`

```
p = re.compile(r"(?P<num>[0-9]+)(?P<str>[a-z]+)")
m = p.search("123456string 67890text")
print(m)
print(m.re.groups, m.re.groupindex)
print(m.re.pattern)
print(m.re.flags)
print(m.string)
print(m.lastindex, m.lastgroup)
print(m.pos, m.endpos)
```

Результат роботи:

```
<_sre.SRE_Match object; span=(0, 12), match='123456string'>
2 {'num': 1, 'str': 2}
(?P<num>[0-9]+)(?P<str>[a-z]+)
32
123456string 67890text
2 str
0 22
```

<об'єкт пошуку>.**group([<id1 або name1> [, ... , <idn або namen>]])**
повертає фрагменти, які відповідають шаблону.

Якщо параметр не заданий або зазначене значення 0, повертається фрагмент, повністю відповідний до шаблону.

Якщо зазначений номер або назва групи, повертається фрагмент, що збігається із цією групою.

Через кому можна вказати декілька номерів або назв груп – у цьому випадку повертається кортеж, який містить фрагменти, що відповідає групам.

Якщо немає групи із зазначеним номером чи назвою, то виконується виключення **Indexerror**.

Приклад 19.

```
import re
p = re.compile(r"(?P<num>[0-9]+)(?P<str>[a-z]+)")
m = p.search("123456string 67890text")
print("Повна відповідність:",m.group(),m.group(0))
print("Доступ по індексу:",m.group(1), m.group(2))
print("Доступ по назві:",m.group('num'), m.group('str'))
print("Декілька параметрів:",m.group(1,2), m.group('num','str')
```

Результат роботи:

Повна відповідність: 123456string 123456string

Доступ по індексу: 123456 string

Доступ по назві: 123456 string

Декілька параметрів: ('123456', 'string') ('123456', 'string')

<об'єкт пошуку>**.groupdict([<Значення за замовчуванням>)** – повертає словник, що містить значення іменованих груп. За допомогою необов'язкового параметра можна вказати значення, яке буде виводитися замість значення None груп, що не мають збігів:

Приклад 20.

```
import re
p = re.compile(r"(?P<num>[0-9]+)(?P<str>[a-z])?")
m = p.search("123456")
print(m.groupdict())
print(m.groupdict(""))
```

Результат роботи:

```
{'num': '123456', 'str': None}
{'num': '123456', 'str': ''}
```

<об'єкт пошуку>**.groups**([<Значення за замовчуванням>])– повертає кортеж, що містить значення всіх груп. За допомогою необов'язкового параметра можна вказати значення, яке буде виводитись замість значення None для груп, що не мають збігів:

Приклад 21.

```
import re
p = re.compile(r"(?P<num>[0-9]+)(?P<str>[a-z])?")
m = p.search("123456")
print(m.groups())
print(m.groups(""))
```

Результат роботи:
('123456', None)
('123456', '')

<об'єкт пошуку>**.start([<Номер або назва групи>])** – повертає індекс початку фрагмента, який відповідає заданій групі. Якщо параметр не зазначений, то фрагментом є повна відповідність з шаблоном. Якщо відповідності немає, то повертається значення -1;

<об'єкт пошуку>**.end([<Номер або назва групи>])** – повертає індекс кінця фрагмента, відповідного заданій групі. Якщо параметр не зазначений, то фрагментом є повна відповідність з шаблоном. Якщо відповідності немає, то повертається значення -1;

<об'єкт пошуку>**.span([<Номер або назва групи>])** – повертає кортеж, що містить початковий і кінцевий індекси фрагмента, відповідного до заданої групи. Якщо параметр не зазначений, то фрагментом є повна відповідність з шаблоном. Якщо відповідності немає, то повертається значення (-1, -1) .

Приклад 22.

```
import re
p = re.compile(r"(?P<num>[0-9]+)(?P<str>[ a-z]+)")
s = "str123456str"
m = p.search(s)
print(m.start(), m.end(), m.span())
print(m.start(1), m.end(1), m.start("num"), m.end("num"))
print(m.start(2), m.end(2), m.start("str"), m.end("str"))
print(m.span(1), m.span("num"), m.span(2), m. span("str"))
print(s[m.start(1):m.end(1)], s[m.start(2):m.end(2)])
```

Результат роботи:

```
3 12 (3, 12)
3 9 3 9
9 12 9 12
(3, 9) (3, 9) (9, 12) (9, 12)
123456 str
```

<об'єкт пошуку>**.expand(<Шаблон>)** – виконує заміну в рядку. Усередині зазначеного шаблону можна використовувати зворотні посилання: **\номер групи** й **\g<номер групи>** і **\g<назва групи>**.

Приклад 23. Замінемо два теги місцями:

```
import re
p = re.compile(r"<(?P<tag1>[a-z]+)><(?P<tag2>[a-z]+)>")
m = p. search("<br><hr>")
print(m.expand(r"<\2><\1>"))
print(m.expand(r"<\g<2>><\g<1>>"))
print(m.expand(r"<\g<tag2>><\g<tag1>>"))
print(m.expand(r"<\g<tag2>><\g<tag1>>"))
```

Результат роботи:

<hr>
<hr>
<hr>
<hr>

Використання функції `search()`

Перевіримо на відповідність шаблону введену користувачем адресу електронної пошти:

Приклад 24.

```
import re
email = input("Введіть e-mail: ")
pe = r"^([a-z0-9_.-]+)@(([a-z0-9-]+\.)+[a-z]{2,6})$"
p = re.compile(pe, re.I | re.S)
m = p.search(email)
if not m: print("E-mail не відповідає шаблону")
else:
    print("E-mail", m.group(0), "відповідає шаблону")
    print ("ящик:", m.group(1), "домен:", m.group(2))
```

Результат виконання:

```
Введіть e-mail: user@ukr.net
E-mail user@ukr.net відповідає шаблону
ящик: user домен: ukr.net
```

Пошук всіх збігів з шаблоном

Формат функції re.**findall** модуля re:

re.findall(<Шаблон>,<Рядок>[,<Модифакатор>])

Повертається список із фрагментами, у протилежному випадку повертається порожній список.
Якщо всередині шаблону є більше, ніж одна група, то кожний елемент списку буде кортежем, а не рядком.

У параметрі <Шаблон> вказується рядок з регулярним вираженням або скомпільоване регулярне вираження.

У параметрі <Модифікатор> можна вказати прапори, используемыев функції compile().

У параметрі рядок<Рядок> задаємо рядок, у якому відбувається пошук.

Приклад 25.

```
import re
re.findall(r"[0-9]+", "1 2 3 4 5 6")
p = re.compile(r"[0-9]+")
print(re.findall(p, "1 2 3 4 5 6"))
```

Результат виконання:

```
['1', '2', '3', '4', '5', '6']
```

Формат методу <об'єкт пошуку>. **findall** модуля re:

<об'єкт пошуку>. **findall**(<Рядок>[,<Початкова позиція>[,<Кінцева позиція>]])

Якщо відповідності знайдені, повертається список із фрагментами, у протилежному випадку повертається порожній список.

Якщо всередині шаблону є більше, ніж одна група, то кожний елемент списку буде кортежем, а не рядком.

Приклад 26.

```
import re
p1 = re.compile(r"[0-9]+")
print(p1.findall("2012, 2013, 2014, 2015, 2016"))
p2 = re.compile(r"[a-z]+")
print(p2.findall("2012, 2013, 2014, 2015, 2016"))
t = r"[0-9]{3}-[0-9]{2}-[0-9]{2}"
p = re.compile(t)
print(p.findall("322-55-98"))
print(p.findall("322-55-98, 678-56-12"))
```

Результат виконання:

```
['2012', '2013', '2014', '2015', '2016']
[]
['322-55-98']
['322-55-98', '678-56-12']
```

Формат функції re.**findall** модуля re:

re.finditer(<Шаблон>,<Рядок>[,<Модифікатор>])

Функція `finditer()` аналогічна функції `findall()`, але повертає ітератор, а не список. На кожній ітерації циклу повертається об'єкт `Match`.

У параметрі <Шаблон> вказується рядок з регулярним виразом або скомпільваний регулярний вираз.

У параметрі <Модифікатор> можна вказати прапори, використовувані в функції `compile()`.

У параметрі рядок<Рядок> задаємо рядок, у якому відбувається пошук.

Приклад 27. Одержимо вміст між тегами:

```
import re
p = re.compile(r"<b>(.+?)</b>", re.I | re.S)
s = "<b>Text1</b>Text2<b>Text3</b>"
for m in re.finditer(p, s):
    print(m.group(1))
```

Результат виконання:

```
Text1
Text3
```

Формат методу <об'єкт пошуку>. **finditer** модуля re:

<об'єкт пошуку>.**finditer**(<Рядок>[,<Початкова позиція>[,<Кінцева позиція>]])

Метод `finditer()` аналогічний методу `findall()`, але повертає ітератор, а не список. На кожній ітерації циклу повертається об'єкт `Match`.

Приклад 28.

```
import re
p = re.compile(r"[0-9]+")
for m in p.finditer("2012, 2013, 2014, 2015, 2016"):
    print(m.group(0), "start:", m.start(), "end:", m.end())
```

Результат виконання:

```
2012 start: 0 end: 4
2013 start: 6 end: 10
2014 start: 12 end: 16
2015 start: 18 end: 22
2016 start: 24 end: 28
```

Заміна в рядку. Метод <об'єкт пошуку>.sub

Метод `sub()` шукає всі збіги із шаблоном і заміняє їхнім зазначеним значенням. Якщо збіги не знайдені, повертається початковий рядок. Метод має наступний формат:

```
sub(<фрагмент або посилання на функцію>,
<Рядок для заміни>[,<Максимальна кількість
замін>])
```

Усередині нового фрагмента можна використовувати зворотні посилання **\номер групи**, **\g<номер групи>** і **\g<назва групи>.**

Приклад 29. Поміняємо два теги місцями:

```
import re
p = re.compile(r"<(?P<tag1>[a-z]+)><(?P<tag2>[a-z]+)>")
print(p.sub(r"<\2><\1>", "<br><hr>")) # \номер
print(p.sub(r"<\g<2>><\g<1>>", "<br><hr>"))# \g<номер>
print(p.sub(r"<\g<tag2>><\g<tag1>>", "<br><hr>")) # \g<назва>
```

Результат виконання:

<hr>
<hr>
<hr>

Як перший параметр можна вказати посилання на функцію.

Приклад 30.

```
import re
def repl(m):
  """ Функція для заміни. m – об'єкт Match """
  x = int(m.group(0))
  x += 10
  return "{0}".format(x)
p = re.compile(r"[0-9]+")
# Заміняємо всі входження
print(p.sub(repl, "2012, 2013, 2014, 2015"))
# Заміняємо тільки перші два входження
print(p.sub(repl, "2012, 2013, 2014, 2015", 2))
```

Результат виконання:

```
2022, 2023, 2024, 2025
2022, 2023, 2014, 2015
```

Функція re.sub()

re.sub(<Шаблон>, <фрагмент або посилання на функцію>, <Рядок для заміни>[,<Максимальна кількість замін> [, flags =0])

Приклад 31. Поміняємо два теги місцями, а також змінимо регістр букв:

```
import re
def repl(m) :
    tag1 = m.group("tag1").upper()
    tag2 = m.group("tag2").upper()
    return "<{0}><{1}>".format(tag2, tag1)
p = r"<(?P<tag1>[a-z]+)><(?P<tag2>[a-z]+)>"
print(re.sub(p, repl, "<br><hr>"))
```

Результат виконання:

```
<HR><BR>
```

Метод <об'єкт пошуку>.subn()

Метод `subn()` аналогічний методу `sub()`, але повертає не рядок, а кортеж із двох елементів: зміненого рядка й кількості зроблених замін.

Метод має наступний формат:

```
<об'єкт пошуку>.subn(<фрагмент або посилання на функцію>, <Рядок для заміни>[, <максимальна кількість замін>] )
```

Замінимо всі числа в рядку на 0:

Приклад 32.

```
import re
p = re.compile(r"[0-9]+")
print(p.subn("0", "2014, 2015, 2016, 2017"))
```

Результат виконання:

```
('0, 0, 0, 0', 4)
```

Функція re.subn()

Формат функції:

```
re.subn(<Шаблон>, <Новий фрагмент або посилання на функцію>,<Рядок для заміни>[,<Максимальна кількість замін>
[, flags=0]])
```

Приклад 33.

```
import re
p = r"201[5]"
print(re.subn(p, "2012", "2014, 2015, 2016, 2017"))
```

Результат виконання:

('2014, 2012, 2016, 2017', 1)

Метод <об'єкт пошуку>.expand()

Для виконання замін також можна використовувати метод `expand()`, підтримуваний объектом `Match`.

Формат методу:

<об'єкт пошуку>.expand(<Шаблон>)

Усередині зазначеного шаблону можна використовувати зворотні посилання: **\номер групи**, **\g<номер групи>** і **\g<назва групи>.**

Приклад 34.

```
import re
p = re.compile(r"<(?P<tag1>[a-z]+)><(?P<tag2>[a-z]+)>")
m = p.search("<br><hr>")
print(m.expand(r"<\2><\1>"))  #\номер
print(m.expand(r"<\g<2>><\g<1>>"))# \g<номер>
print(m.expand(r"<\g<tag2>><\g<tag1>>")) # \g<назва>
```

Результат виконання:

```
<hr><br>    <hr><br>     <hr><br>
```

Метод <об'єкт пошуку>.split()

Метод `split()` розбиває рядок по шаблону й повертає список підрядків. Його формат:

```
<об'єкт пошуку>.split(<Початковий рядок>[, <Ліміт>])
```

Якщо в другому параметрі `<Ліміт>` задане число, то в списку опиниться зазначена кількість підрядків.

Якщо підрядків більше від зазначеної кількості, то список буде містити ще один елемент – із залишком рядка.

Приклад 35.

```
import re
p = re.compile(r"[\s,.]+")
print(p.split("word1, word2\nword3\r\nword4.word5"))
print(p.split("wordl, word2\nword3\r\nword4.word5", 2))
```

Результат виконання:

```
['word1', 'word2', 'word3', 'word4', 'word5']
['wordl', 'word2', 'word3\r\nword4.word5']
```

Якщо роздільник у рядку не знайдений, то список буде складатись тільки з одного елемента, що містить початковий рядок:

Приклад 36.

```
import re
p = re.compile(r"[0-9]+")
print(p.split("word, word\nword"))
```

Результат виконання: `['word, word\nword']`

Функція re.split()

Формат функції:

re.split(<Шаблон>, <Початковий рядок>[, <Ліміт>[, flags=0]])

риклад 37.

```
import re
p = re.compile(r"[\s,.]+")
print(re.split(p, "word1, word2\nword3"))
print(re.split(r"[\s,.]+", "word1, word2\nword3"))
```

Результат виконання:

```
['word1', 'word2', 'word3']
['word1', 'word2', 'word3']
```

Функція `re.escape()`

Функція `escape(<Рядок>)` дозволяє екранувати всі спеціальні символи в рядку, отриманому від користувача. Цей рядок надалі можна безпечно використовувати всередині регулярного виразу.

Приклад 38.

```
import re
print(re.escape(r"[]().*"))
```

Результат виконання:

```
\[\]\(\)\.\*
```

Функція `re.purge()`

Функція `purge()` виконує очищення кеша, у якому зберігаються проміжні дані, використовувані в процесі виконання регулярних виразів. Її рекомендують викликати після обробки великої кількості регулярних виразів. Ця функція не повертає ніякого результату.

Приклад 39.

```
import re
re.purge()
```