


Tema 16.

Expresii regulate în

Python



Ce ne așteaptă?

1. **Esența expresiilor regulate**
2. **Tehnici de creare a șabloanelor**
3. **Funcții ale expresiilor regulate**

1. Ce reprezintă expresiile regulate?
2. Unde și când se utilizează expresiile regulate?
3. Ce modul se utilizează pentru lucru cu expresiile regulate?
4. Care este rolul funcțiilor `compile()` și `finditer()` ?
5. Care sunt metodele obiectelor `Match`?
6. Cum se creează șabloanele cu ajutorul claselor de caractere?
7. Care sunt cuantificatorii utilizați la crearea șabloanelor?
8. Care este rolul simbolurilor `^` și `$` la crearea șabloanelor
9. Care este rolul funcțiilor `match()`, `fullmatch()` și `search()`?
10. Care este rolul funcțiilor `findall()` și `split()` ?
11. Care este rolul funcțiilor `sub()` și `subn()` ?

1. Esența expresiilor regulate

Noțiuni de expresii regulate

- **Expresie regulată (Regular Expression – ReGex)** - o secvență de caractere care definește un șablon de căutare
- **Expresia regulată este un mecanism declarativ de reprezentare a unui grup de caractere în conformitate cu un anumit șablon**
- **Expresiile regulate se utilizează atunci când se dorește ca un anumit grup de stringuri să respecte un anumit șablon**
- **Exemple de utilizare a expresiilor regulate:**
 - Expresii regulate pentru scrierea unui număr de telefon
 - Expresii regulate pentru reprezentarea unei adrese de e-mail

Utilizarea expresiilor regulate

- Pentru a dezvolta unele framework-uri de validare
- Pentru a dezvolta aplicații de coincidere a șabloanelor (ctrl-f)
- Pentru a dezvolta traductori precum compilatoare, interpretoare etc
- Pentru a dezvolta circuite digitale
- Pentru a dezvolta protocoale de comunicare precum TCP/IP, UDP etc

Utilizarea modulului re

- Pentru lucru cu expresiile regulate în Python se utilizează modulul re care trebuie importat
- Modulul re conține metode încorporate care permit aplicare expresiilor regulate
- Funcția `compile()` – permite crearea unui obiect `RegexObject` din șablonul specificat:

`obiect_sablon=re.compile("<șablon>")`

- Metoda `finditer()` – permite căutarea șablonului într-un string și generarea unui obiect `Match` de tip iterație ce conține informație despre coincidența șablonului cu conținutul stringului

`obiect_match= obiect_sablon.finditer(stringul de căutare)`

- Funcția `finditer()` permite și crearea `RegexObject` print trecerea șabonului ca parametru

`obiect_match= re.finditer("<șablon>" , stringul de căutare)`

Metode ale obiectelor Match

- Metoda `group()` – permite returnarea șablonului în format string

```
obiect_match= re.finditer("<șablon>", stringul de căutare)  
for i in obiect_match:  
    print(i.group())
```

- Metoda `start()` – permite determinarea indexului din stringul de căutare unde s-a depistat începutul coincidenței cu șablonul

```
obiect_match= re.finditer("<șablon>", stringul de căutare)  
for i in obiect_match:  
    print(i.start())
```

- Metoda `end()` – permite determinarea indexului din stringul de căutare unde șablonul nu mai coincide (ultimul index care coincide +1)

```
obiect_match= re.finditer("<șablon>", stringul de căutare)  
for i in obiect_match:  
    print(i.end())
```

Exemple de utilizare a expresiilor regulate

- **Exemplu cu utilizarea funcției `compile()` pentru crearea obiectului `RegexObject`**

```
import re
count=0
model=re.compile("ab")
coincidente=model.finditer("abaababa")
for c in coincidente:
    count+=1
    print(f"Modelul '{c.group()}' coincide de la indexul {c.start()} pina la indexul {c.end()}")
    print("Numarul de coincidente a modelului cautat in text: ", count)
    print()
```

- **Exemplu cu utilizarea funcției `finditer()` pentru crearea obiectului `RegexObject`**

```
import re
count=0
coincidente=re.finditer("ab","abaababa")
for c in coincidente:
    count+=1
    print(f"Modelul '{c.group()}' coincide de la indexul {c.start()} pina la indexul {c.end()}")
    print("Numarul de coincidente a modelului cautat in text: ", count)
    print()
```


2. Tehnici de creare a șabloanelor

Clasificarea tehnicilor de crearea a șabloanelor

- **Coincidență perfectă** – șablonul reprezintă un string iar în procesul de căutare se cerceta prezența unei coincidențe perfecte

```
coincidente=re.finditer("ab", "abaababa")
```

- **Clase standard de caractere** – se utilizează pentru căutarea unui grup de caractere iar șablonul reprezintă un string cu paranteze pătrate ce respectă anumite condiții

```
coincidente=re.finditer("[abc]", "a39b8k7")
```

- **Clase predefinite de caractere** – se utilizează pentru căutarea unui grup de caractere iar șablonul reprezintă un string ce începe cu simbolul “\” urmat de o literă

```
coincidente=re.finditer("\d", "a39b8k7")
```

- **Cuantificatori** se utilizează pentru a specifica numărul de apariții a unui caracter/grup

```
coincidente=re.finditer("ab", "abaababa")
```

Clase standard de caractere

- Se utilizează pentru căutarea unui grup de caractere iar șablonul reprezintă un string cu paranteze pătrate ce respectă anumite condiții:

[abc] : fie a ori b ori c

[^abc] : toate exceptând a, b și c

[a-z] : toate minusculele simbolurilor alfabetului

[A-Z] : toate majusculele simbolurilor alfabetului

[a-zA-Z] : toate simbolurilor alfabetului

[^a-zA-Z] : toate exceptând simbolurilor alfabetului

[0-9] : toate cifrele de la 0 la 9

[^0-9] : toate exceptând cifrele de la 0 la 9

[a-zA-Z0-9] : toate simbolurilor alfa-numerice

[^a-zA-Z0-9] : toate exceptând simbolurilor alfa-numerice (simboluri speciale)

Exemplu de creare a șabloanelor cu clase standard de caractere

```
import re

coincidenta=re.finditer("[abc]", "a39@8k7M")
# coincidenta=re.finditer("[^abc]", "a39@8k7M")
# coincidenta=re.finditer("[a-z]", "a39@8k7M")
# coincidenta=re.finditer("[a-d]", "a39@8k7M")
# coincidenta=re.finditer("[A-Z]", "a39@8k7M")
# coincidenta=re.finditer("[a-zA-Z]", "a39@8k7M")
# coincidenta=re.finditer("[^a-zA-Z]", "a39@8k7M")
# coincidenta=re.finditer("[0-9]", "a39@8k7M")
# coincidenta=re.finditer("[2-7]", "a39@8k7M")
# coincidenta=re.finditer("[^0-9]", "a39@8k7M")
# coincidenta=re.finditer("[a-zA-Z0-9]", "a39@8k7M")
# coincidenta=re.finditer("[^a-zA-Z0-9]", "a39@8k7M")

for c in coincidenta :
    print(c.group(), "...", c.start())
```

Clase predefinite de caractere

- se utilizează pentru căutarea unui grup de caractere iar șablonul reprezintă un string ce începe cu simbolul “\” urmat de o literă cu o anumită semnificație:

\s : caracterul spațiu

\S : orice caracter exceptând caracterul spațiu

\d : orice cifră de la 0 la 9 [0-9]

\D : orice caracter exceptând cifrele de la 0 la 9 [^0-9]

\w : orice caracter afla-numeric [a-zA-Z0-9]

\W : orice caracter exceptând caracterele afla-numerice [^a-zA-Z0-9]

\+ : doar simbolul special (simbolul + in acest exemplu)

. : orice caracter inclusiv caracterele speciale

Exemplu de creare a șabloanelor cu clase predefinite de caractere

```
import re

coincidenta=re.finditer("\s","a39@+ 8k7M")
# coincidenta=re.finditer("\S","a39@+ 8k7M")
# coincidenta=re.finditer("\d","a39@+ 8k7M")
# coincidenta=re.finditer("\D","a39@+ 8k7M")
# coincidenta=re.finditer("\w","a39@+ 8k7M")
# coincidenta=re.finditer("\W","a39@+ 8k7M")
# coincidenta=re.finditer("\+", "a39@+ 8k7M")

for c in coincidenta :
    print(c.group() , "...",c.start())
```

Cuantificatori la crearea șabloanelor

- Se utilizează pentru a specifica numărul de apariții a unui caracter/grup, iar șablonul reprezintă un string ce include caracterul/grupul specificat urmat de caractere cu o anumită semnificație:

a : simbolul a exact o singură dată (doar 1)

a+ : simbolul a cel puțin o singură dată (1 sau mai mult)

a* : simbolul a de oricare număr inclusiv și de 0 (0, 1 sau mai mult)

a? : simbolul a cel mult o singură dată inclusiv și 0 (0 sau 1)

a{m} : simbolul a de exact m ori (doar m)

a{m,n} : simbolul a de cel puțin m ori și cel mult n ori (de la m până la n)

Exemplu de creare a șabloanelor cu cuantificatori

```
import re

coincidenta=re.finditer("a","abaacdadae")
# coincidenta=re.finditer("a+","abaacdadae")
# coincidenta=re.finditer("a*","abaacdadae")
# coincidenta=re.finditer("a?","abaacdadae")
# coincidenta=re.finditer("a{3}","abaacdadae")
# coincidenta=re.finditer("a{2,3}","abaacdadae")

for c in coincidenta :
    print(c.group() ,"... ",c.start())
```

Simbolurile ^ și \$ la crearea șablonelor

- Simbolul **^** permite verificarea prezenței șablonului la începutul stringului de căutare

```
import re
coincidenta=re.finditer("^a","abaacdadae")
for c in coincidenta :
    print(c.group(),"...",c.start())
```

- Simbolul **\$** permite verificarea prezenței șablonului la sfârșitul stringului de căutare

```
import re
coincidenta=re.finditer("a$","abaacdada")
for c in coincidenta :
    print(c.group(),"...",c.start())
```


3. Funcții ale expresiilor regulate

Funcția match()

- Permite crearea unui obiect Match dacă stringul de căutare începe cu stringul șablonului sau returnează None în caz contrar

obiect_match= re.match("<șablon>" , stringul de căutare)

- Exemplu de utilizare:

```
import re
m=re.match("bc", "abcabdefg")
if m != None:
    print('Stringul de cautare incepe cu valoarea sablonului')
else:
    print('Stringul de cautare nu incepe cu valoarea sablonului')
```

Funcția `fullmatch()`

- Permite crearea unui obiect `Match` dacă stringul de căutare coincide în totalitate stringul șablonului sau returnează `None` în caz contrar

`obiect_match = re.fullmatch("<șablon>", stringul de căutare)`

- Exemplu de utilizare:

```
import re
m=re.fullmatch("abcabdefg", "abcabdefg")
if m != None:
    print('Stringul de cautare coincide cu valoarea sablonului')
else:
    print('Stringul de cautare nu coincide cu valoarea sablonului')
```

Funcția search()

- Permite crearea unui obiect Match cu prima coincidență dacă în stringul de căutare se găsește șablonul sau returnează None în caz contrar

obiect_match= re.search("<șablon>", stringul de căutare)

- Exemplu de utilizare:

```
import re
m=re.search("ba", "abaaaba")
if m != None:
    print(f'Sablonul a fost gasit in string la indexul {m.start()}')
else:
    print('Sablonul nu a fost gasit in string')
```

Funcțiile findall() și split()

- Funcția findall() - permite crearea unei liste cu stringurile șabloanelor care coincid în stringul de căutare

lista= re.findall("<șablon>", stringul de căutare)

- Exemplu de utilizare am metodei:

```
import re
l=re.findall("[a-z0-9]", "a39@+ 8k7M")
print(l)
```

- Funcția split() - permite divizarea stringul de căutare conform după criteriul stringului șablonului și returnează lista cu substringuri

lista= re.split("<șablon>", stringul de căutare)

- Exemplu de utilizare:

```
import re
l=re.split("[0-9]", "a3bc9@+ 8k7M")
print(l)
```

Funcțiile sub() și subn()

- Funcția sub() - permite substituirea stringurilor șabloanelor care coincid în stringul de căutare cu un substring de înlocuire

string_nou= re.sub("<șablon>", substring de înlocuire, stringul de căutare)

- Exemplu de utilizare am metodei:

```
import re
s=re.sub("\W","#", "a3bc9@+ 8k7M")
print(s)
```

- Funcția subn() – exact ca și sub() doar că returnează un tuplu cu 2 elemente: noul string și numărul de substituții

string_nou= re.subn("<șablon>", substring de înlocuire, stringul de căutare)

- Exemplu de utilizare:

```
import re
s=re.subn("\W","#", "a3bc9@+ 8k7M")
print(s)
```

Exemple de utilizarea a expresiilor regulate

- Validarea unui număr de telefon din Republica Moldova

```
import re
def validare(string):
    c = re.fullmatch("0[267][0-9]{7}",string)
    if c!=None:
        print("Numar de telefon valid")
    else:
        print("Numar de telefon nevalid")
validare('079890874')
```

- Validarea expresiei unei operații aritmetice ce ar utiliza simbolurile: +, -, *, /, % și ^

```
import re
def validare(string):
    sablon="[0-9]+[\\+\\-\\*\\/\\%\\*\\^][0-9]+"
    c = re.fullmatch(sablon,string)
    if c!=None:
        print("Expresie valida")
    else:
        print("Expresie nevalida")
validare('123/232')
```

Expresii regulate la validarea adreselor de email

- Validarea adresei de email corporativ UTM

```
import re
def validare(string):
    sablon="[a-z]+\.[a-z]+@[a-z]{2,3}\.utm\.md"
    c = re.fullmatch(sablon,string)
    if c!=None:
        print("Adresa de email valida")
    else:
        print("Adresa de email nevalida")
validare('pavel.nicolaev@tlc.utm.md')
```

- Validarea oricărei adresei de email

```
import re
def validare(string):
    sablon="[a-zA-Z0-9_\.]+\@[a-z\.\.][a-z]{2,3}"
    c = re.fullmatch(sablon,string)
    if c!=None:
        print("Adresa de email valida")
    else:
        print("Adresa de email nevalida")
validare('nicolaev.pavel@gmail.com')
```