Formal Languages and Translators

Lab 1. Regular Expressions (REGEX)

Laborator LFT (Limbaje Formale și Translatoare)

- Contact laborant:
  - Email: stefanpopescu923@gmail.com
  - Teams:
    - Team LFT: 1v8yqc6
    - Mesaj privat: Ovidiu Stefan Popescu
- Materiale (Curs + Laborator) - Moodle:
  - Link: https://moodle.cs.utcluj.ro/course/view.php?id=631
  - Parola: lex&Yacc6

1

WEEK 1-3

REGEX + Lex

Învățăm despre expresii regulate și Lex (generator de analizator lexical)

2

WEEK 4

Test Lex

Test pe Moodle din expresii regulate și Lex: quiz + răspuns scurt

3

WEEK 4-8

Yacc + Lex

Învățăm despre Lex și Yacc (parser generator)

4

WEEK 9

Test Yacc

Test pe Moodle din Yacc și Lex: quiz + probleme

5

WEEK 10-13

Proiect + Alte tools

Veți realiza un proiect singuri sau în echipă. Vom studia alte tool-uri relevante LFT.

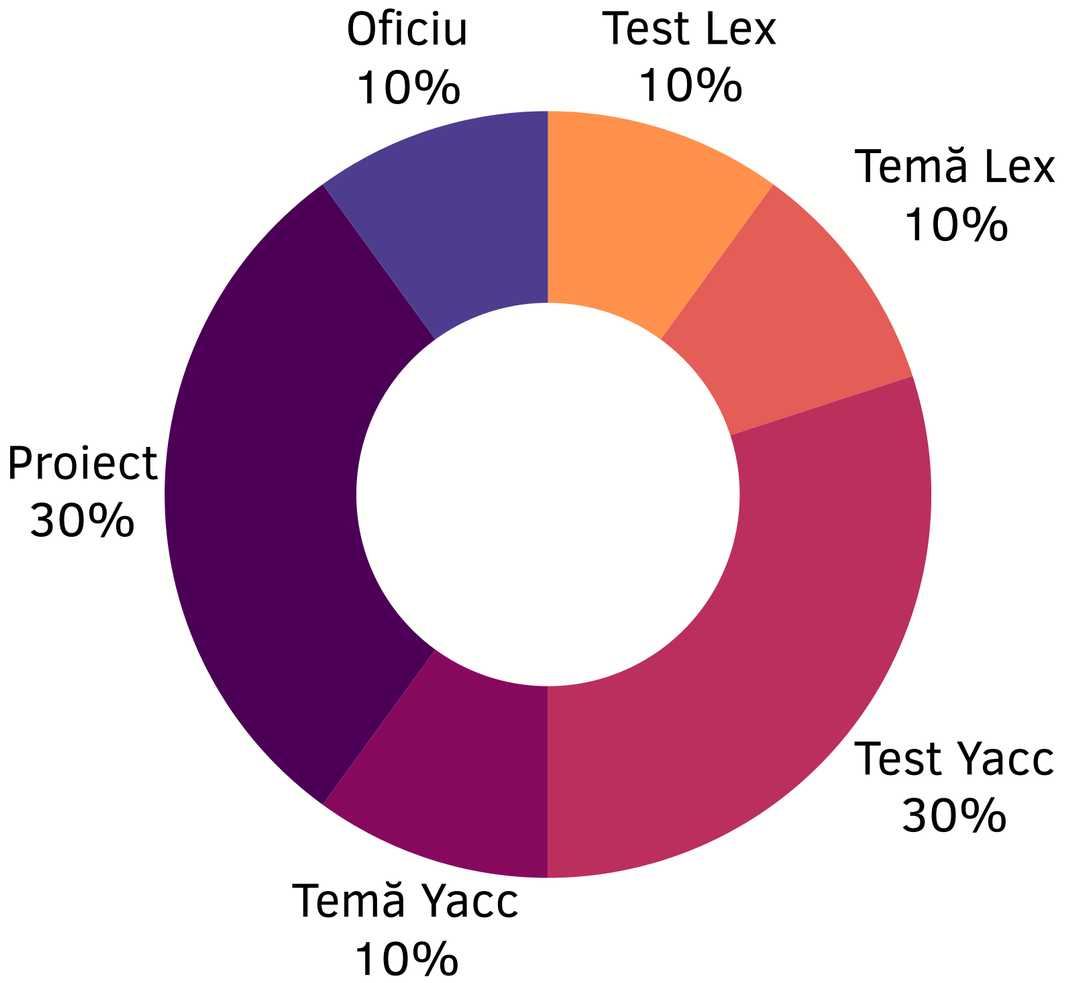
6

WEEK 14

Prezentare proiect

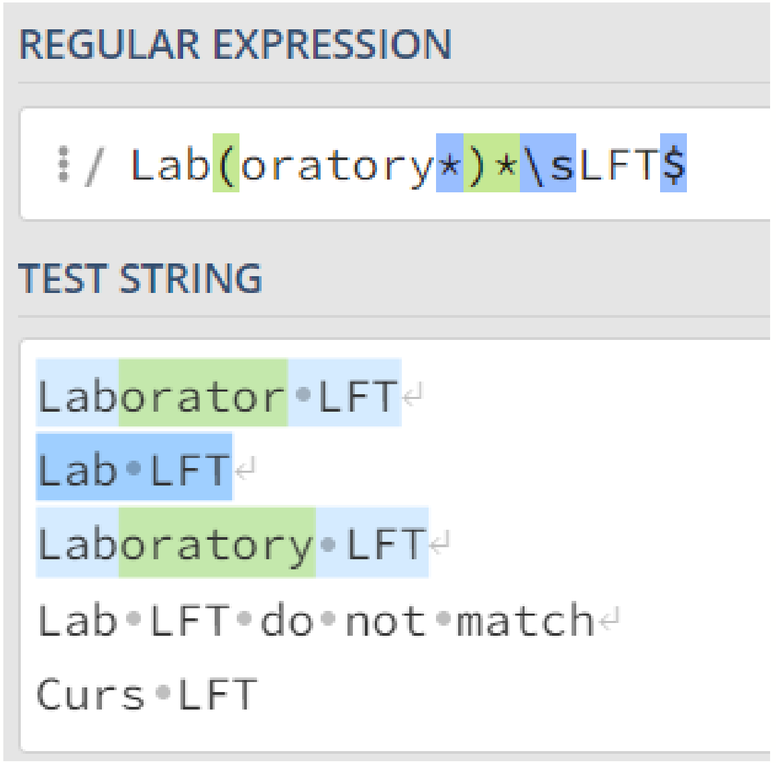
Evaluare

- Notă finală = 40% * Laborator + 60% * Curs
- Copiatul se penalizează cu nota 1.
- Puncte bonus:
  - activitate la laborator (rezolvarea cerințelor)
  - temă bonus în săptămânile 10-13



REGular EXpressions

- Ne permit să căutăm anumite patterns în text.
- Avem librării pentru regex în majoritatea limbajelor:
  - Python: **re**
  - Java: **java.util.regex**
- Pot fi studiate, înțelese și aplicate în mod independent de alte domenii sau tehnologii.



Componente REGEX

1. Caractere simple
2. Clase de caractere
3. Clase predefinite de caractere
4. Caractere non-printable
5. Cuantificatori
6. Grupuri și operatorul ‘|’

Componente REGEX

1. Caractere simple

- Sunt caractere luate ca atare.
- Exemple:
  - **/a/** va căuta “a” în text
  - **/laborator/** va căuta “laborator” în text
  - **/fisier\.exe/** va căuta “fisier.exe” în text

Componente REGEX

2. Clase de caractere

- Grupuri de caractere: [aeiou] - grup de vocale
- Intervale de caractere: [a-Z], [0-9], [a-f1-3]
- Intervale inverse: [^4-8] - face match la caracterele care nu sunt in intervalul [4, 8].

- Exemplu: **/[Ll]aborator/** va găsi:
  - Laborator
  - laborator

Componente REGEX

3. Clase predefinite de caractere

- \d -> [0-9]
- \w -> [a-zA-Z0-9_]
- \s -> spațiu, \t, \n
- ‘.’ -> orice caracter

\D -> [^0-9]

\W -> [^a-zA-Z0-9_]

\S -> Fara spațiu, \t, \n

Componente REGEX

4. Caractere non-printable

- \t - tab
- \r - return
- \n - new line
- \b - delimitare de cuvânt
- \B - nu este delimitare ce cuvânt
- ^ - început de string
- $ - sfârșit de string

Componente REGEX

5. Cuantificatori

- {min, max}: /a{3, 6}/ caută între 3 și 6 caractere ‘a’
- {min,}: /a{3,}/ caută minim 3 caractere ‘a’
- {,max}: /a{,3}/ caută maxim 3 caractere ‘a’
- {exact}: /a{3}/ caută exact 3 caractere ‘a’
- ‘?’ -> {0,1} - o apariție sau niciuna
- ‘*’ -> {0,} - minim o apariție sau niciuna
- ‘+’ -> {1,} - minim o aparitie

Componente REGEX

6. Grupuri

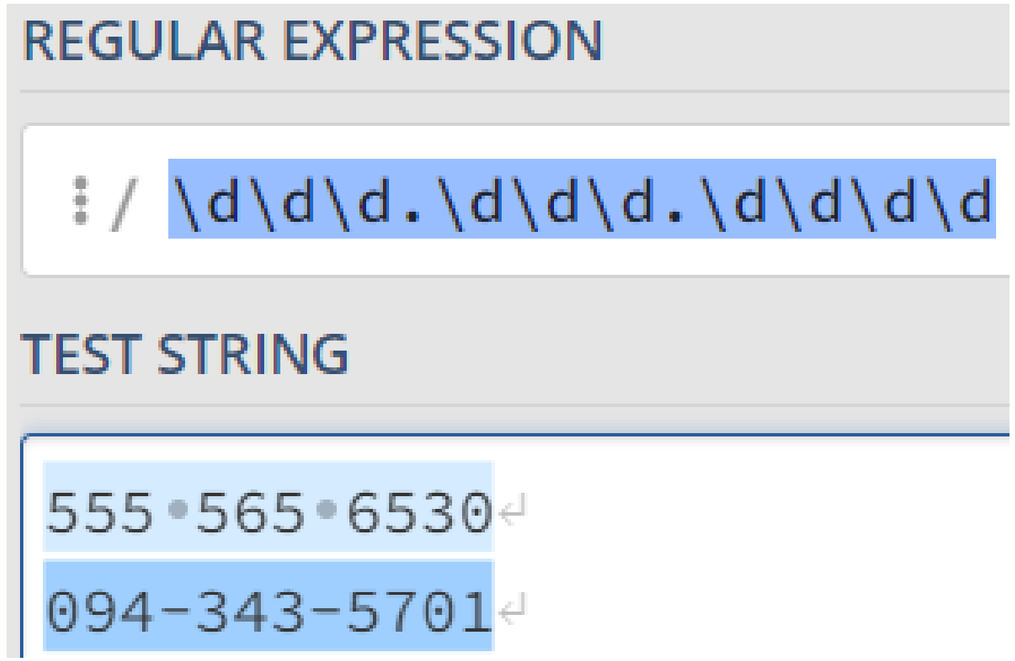
- ‘(’ și ‘)’ - definesc grupuri
- ‘|’ - operatorul SAU

- Exemplu: **/lab(oratory|orator|)/** caută:
  - laboratory
  - laborator
  - lab

Exemple practice

Căutarea numerelor de telefon:

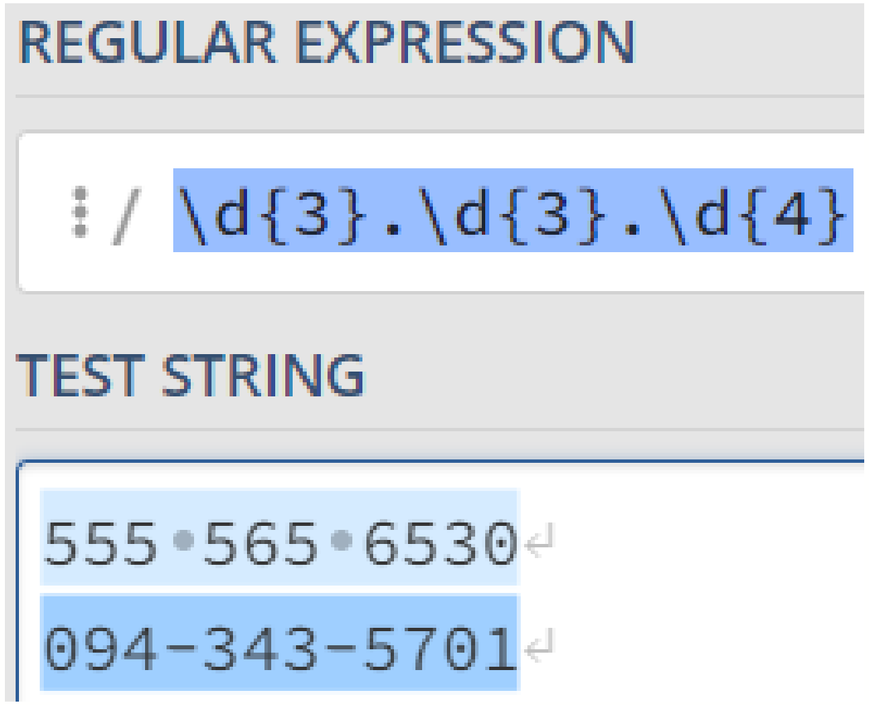
- '**\d**' ține locul unei cifre
- '**.**' ține locul oricărui caracter



Exemple practice

Căutarea numerelor de telefon:

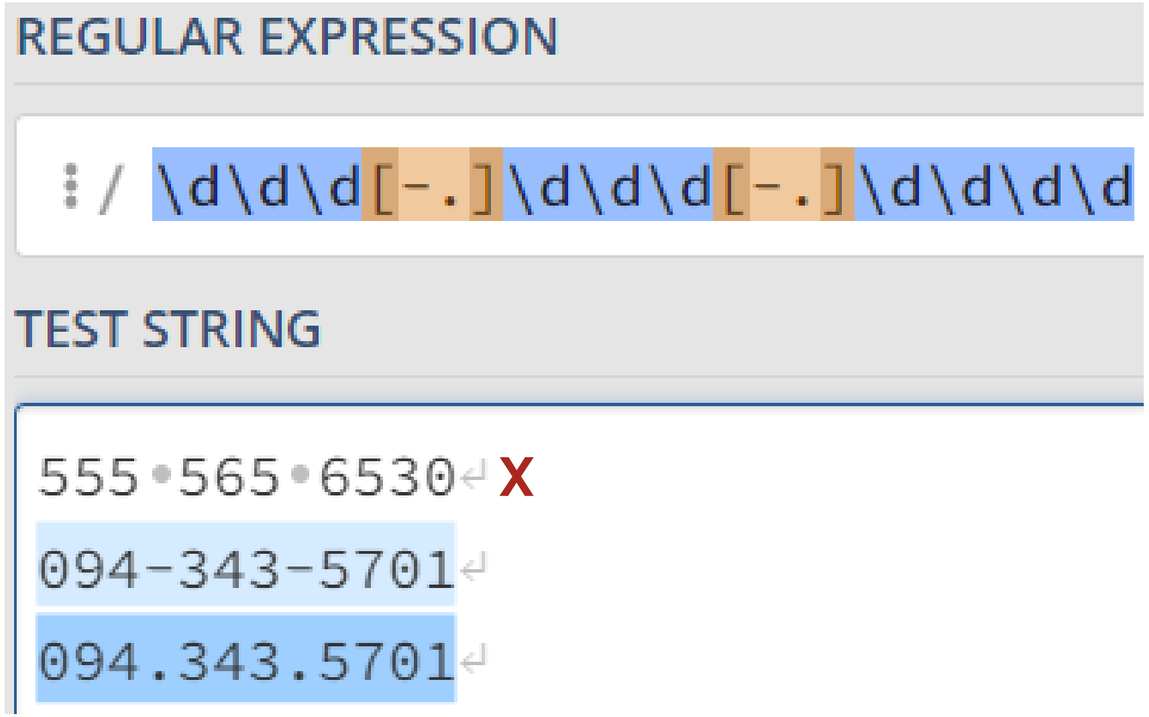
- '**\d**' ține locul unei cifre
- '**\d{3}**' ține locul a trei cifre
- '**.**' ține locul oricărui caracter



Exemple practice

Căutarea numerelor de telefon:

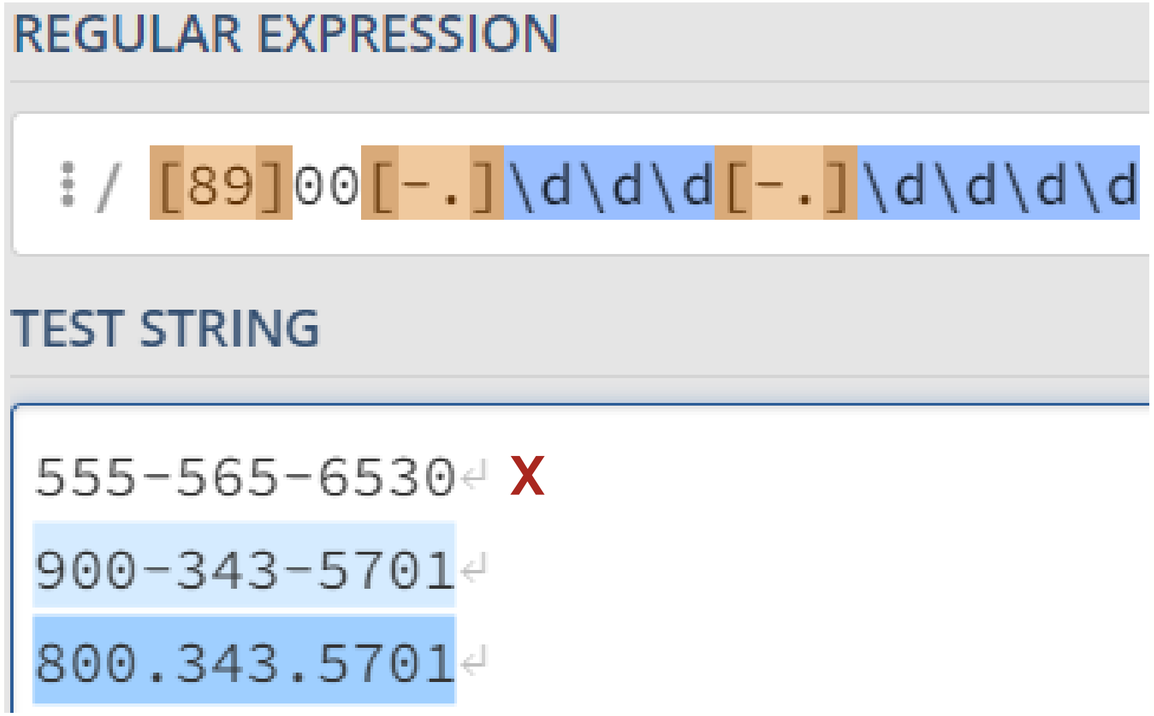
- '**\d**' ține locul unei cifre
- '**[-.]**' ține locul unui caracter '**-**' sau '**.**'



Exemple practice

Căutarea numerelor de telefon:

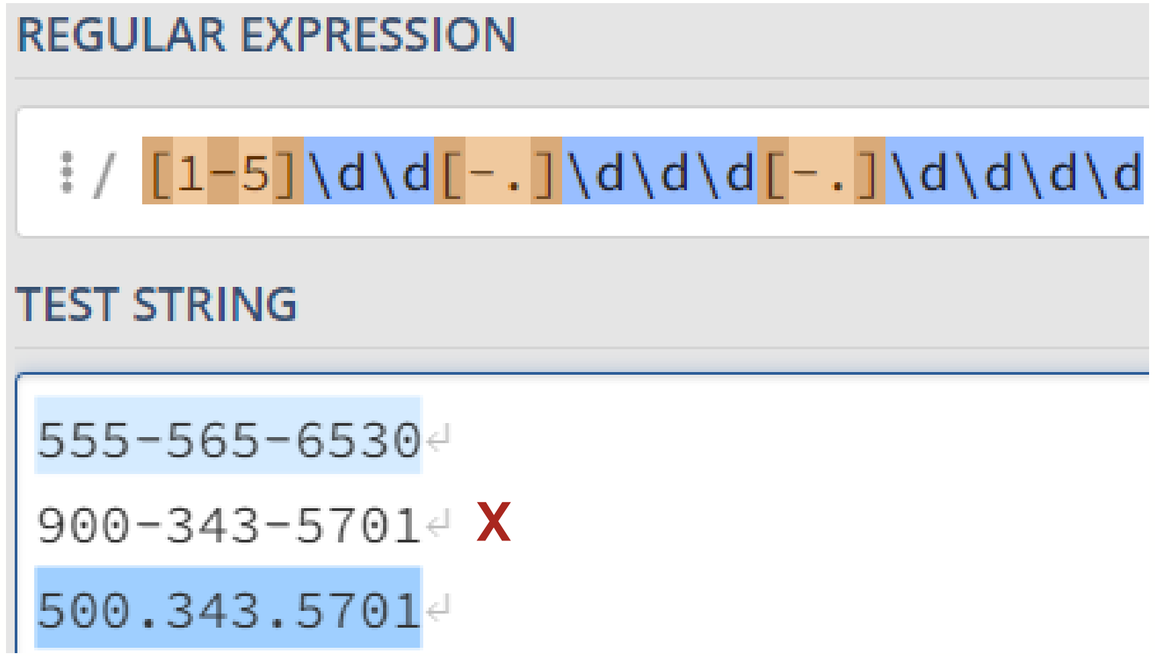
- ’**\d**’ ține locul unei cifre
- ’**[-.]**’ ține locul unui caracter ‘**-**’ sau ‘**.**’
- ‘**[89]**’ ține locul cifrelor 8 sau 9



Exemple practice

Căutarea numerelor de telefon:

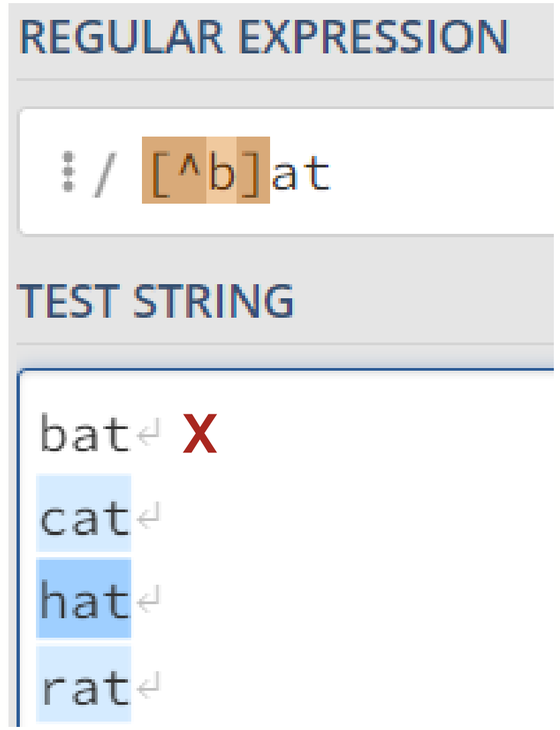
- ’**\d**’ ține locul unei cifre
- ’**[-.]**’ ține locul unui caracter ‘**-**’ sau ‘**.**’
- ‘**[1-5]**’ ține locul oricărei cifre de la 1 la 5



Exemple practice

Căutarea tuturor cuvintelor de forma “*at”, exceptând cuvântul “bat”:

- ’**^**’ transformă clasa în blacklist
- ’**[^b]**’ ține locul oricărei litere, în afară de litera ‘b’



REGEX in Python

```
import re

text_to_search = "abcdefghi...abchello"
pattern = re.compile(r"abc")
matches = pattern.finditer(text_to_search)

for match in matches:
  print(match)
```

Cod executat

Output

```
<re.Match object; span=(0, 3), match='abc'>
<re.Match object; span=(12, 15), match='abc'>
```