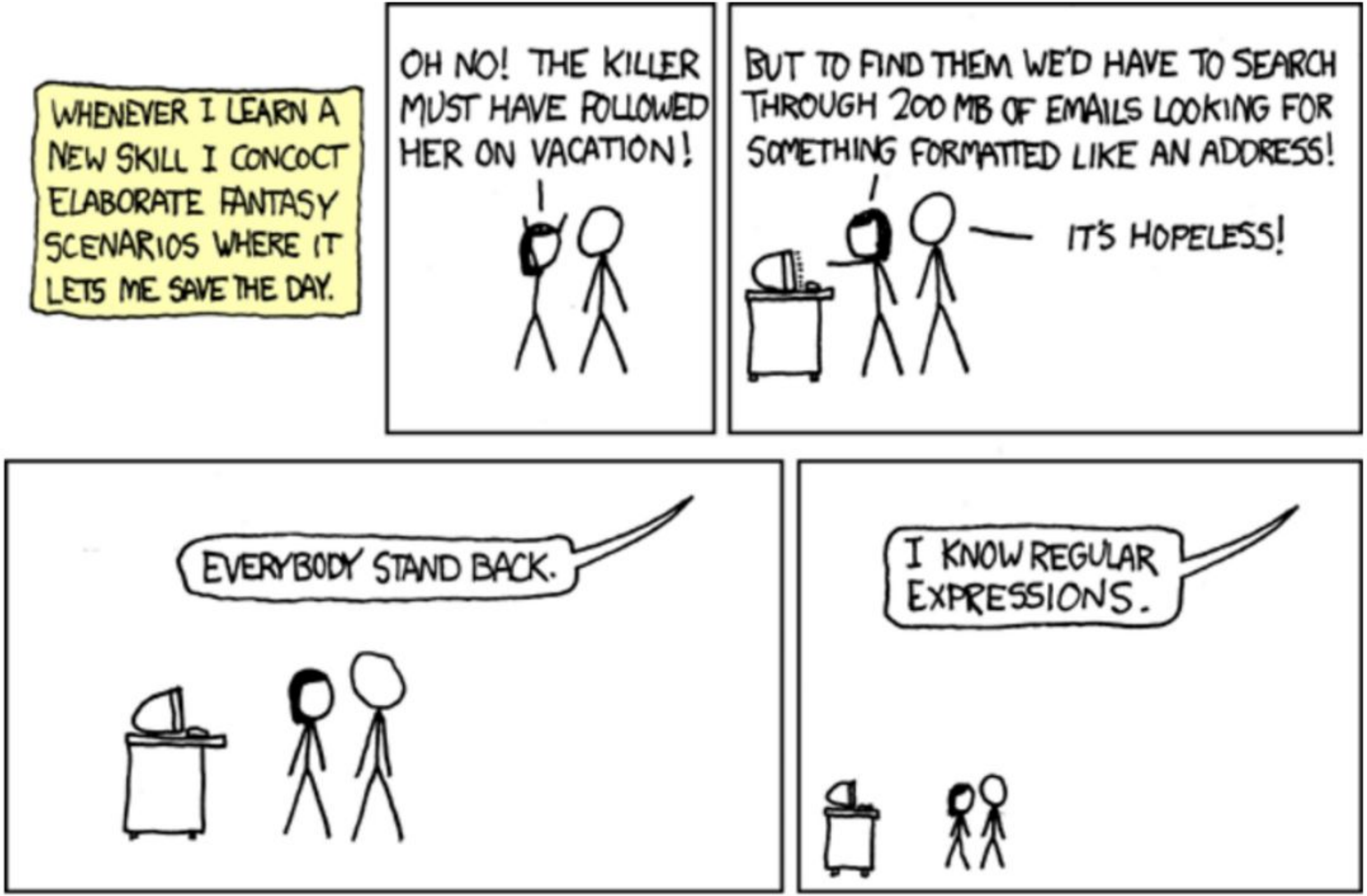
Processamento de Linguagem Natural

Luís Filipe da Costa Cunha
lfc@di.uminho.pt

José João Almeida
jj@di.uminho.pt


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.

Expressões Regulares

“Regular expressions are extremely useful in extracting information from text such as code, log files, spreadsheets, or even documents.”

- **regular expression** ("regex"): describes a pattern of text
  - can test whether a string matches the expr's pattern
  - can use a regex to search/replace characters in a string
  - very powerful, but tough to read

- regular expressions occur in many places:
  - text editors (TextPad) allow regexes in search/replace
  - languages: JavaScript; Java `Scanner`, `String split`
  - Unix/Linux/Mac shell commands (`grep`, `sed`, `find`, etc.)

Regular Expressions

How can we search for any of there?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks

regex to the rescue!
[wW]oodchuck

web.stanford.edu

Disjunction and Intervals

[AEIOU]	any uppercase vowel
[12345678]	any digit
alun[oa]	aluno, aluna
[A-Z]	any uppercase letter between A and Z
[a-z]	cany lowercase letter between a and z
[0-9]	any digit between 0 and 9
[a-zA-Z0-9]	any letter or digit
[^aeiou]	not a lowercase vowel
[^Ss]	...
[^e^]	...
a^b	...

Expressões Regulares

```
import re
text: "Bruno loves programming in Python. 2022 will be a great year!!"
```

```
regex: 'Bruno'
match:
```

```
regex: 'Brun[oa]'
match:
```

```
regex: [0-9][0-9][0-9][0-9]
match:
```

```
regex: '[A-Z][a-z][a-z][a-z][a-z]'
match:
```

Character Classes

- \d Digit ([0-9])
- \D not \d
- \w letter digit or underscore ([a-zA-Z0-9_])
- \W not \w
- \s whitespace
- \S not whitespace

```
text: "Bruno loves programming in python. 2022 will be a great year!!"

pattern:
match: 2022
```

Anchors

- ^ beginning of line
- $ end of the line
- \b word boundary
- \B not word boundary

text: "Bruno loves programming in python!! 2023 will be a great year to create a program !!"

regex: ^\d\d\d\d	regex:	regex: program	regex:
match:	match: !! (only at the end)	match:	match: program (word)

Quantifiers

`*`	`0 or more times`
`+`	`1 or more times`
`?`	`0 or 1 times.`
`{n}`	`exactly n occurences`
`{n,}`	`n or more occurences`
`{n, m}`	`between n and m occurences`

Quantifiers

```
import re
```

Text:	Pattern:	Result:
Is this a color or colour?	'colou?r+'	'color', 'colour'
The class started at February 1, 2020	'[0-9]+'	'1', '2020'
Javascript is not Java	Java[a-zA-Z]*	'Javascript', 'Java'
2023 will be a great year!!!	[0-9]{2,}	'2023'
University of Minho is a great place to learn!	\b[a-z]{1,3}\b	'of', 'is' 'a' 'to'

Special characters

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

- You can escape them by prefixing a backslash

Solve the woodchuck problem!!

```
Woodchucks is  another name for groundhog!
groundhog|woodchuck
```

web.stanford.edu

Regex Functions

- **match** - Try to apply the pattern at the start of the string
- **search** - Scan through string looking for the first location where thes regular expression produces a match
- **findall** - Return a list of all non-overlapping matches in the string
- **sub** - Replace occurrences of the regex pattern
- **split** - Split the source string by the occurrences of the pattern

search match findall

```
re.match(r'...','02-03-2022, esta linha começa com uma data')
re.match(r'\d{2}-\d{2}-\d{4}','O Carnaval foi no dia 01-03-2022')


re.search(r'\d{2}-\d{2}-\d{4}','O Carnaval foi no dia 01-03-2022')
re.search(r'\d{2}-\d{2}-\d{4}','O Carnaval foi no dia 01-03-2022 e a Páscoa é dia 17-04-2022')

re.findall(r'\d{2}-\d{2}-\d{4}','O Carnaval foi no dia 01-03-2022 e a Páscoa é dia 17-04-2022')
```

Raw String

A raw string considers backslashes as literal characters.

```
text ="Hello,\nI'm a student"
print(text)
Output:

  Hello,

  I'm a student
```

```
text =r"Hello,\nI'm a student"
print(text)
Output:

  Hello,\nI'm a student
```

Strings in python can be represented in multiple ways

```
len("\n") #1            len("\\n") #2            len(r"\n") #2
```

If you want a Python regular expression object which matches a newline character, then you need a 2-character string, consisting of the backslash character followed by the n character

[7.2. re — Regular expression operations — Python 2.7.18 documentation]

Capturing groups

- Capturing groups are a way to treat multiple characters as a single unit

```
re.findall(r'(Sra|Sr|Senhora|Senhor)','A Senhora Teresa encontrou a Sra. Maria no shopping.' )
['Senhora', 'Sra']
```

```
re.search(r'alde(ão|ãe|õe)s','Os aldeãos fizeram uma festa na aldeia' )
```

```
re.search(...,'<span>This is the span content<\span> )
```

- Operators after a capturing group are applied to the whole group pattern: `re.match(r'(go)+','gogogogo now!')`

split, sub

```
re.split(r' ','O Carnaval foi no dia 01-03-2022 e a Páscoa é dia 17-04-2022')
['O', 'Carnaval', 'foi', 'no', 'dia', '01-03-2022', 'e', 'a', 'Páscoa', 'é', …]

re.sub(r'and', '&', 'And Baked Beans and Spam' )


re.sub(r'and', '&', 'And Baked Beans and Spam' ,flags=re.IGNORECASE)
```

Watch Out for The Greediness!

- Use regex to match an HTML tag of the following text:

  `<span> Hello Wolrd! <\span>`

  regex: `<.+>`

  result: `<span> Hello Wolrd! <\span>`

- Greedy will consume as much as possible
- Making it lazy (non greedy)!

  regex: `<.+?>`

  result: `<span>`

Errors

Suppose you want to find all the occurrences of the word 'the' in a given text.

Pattern 1: `re.search(r'the',text)`

Error 1: Not matching things that we should have matched (The)

Pattern 2: `re.search(r'[Tt]he',text)`

Error 2: Matching strings that we should not have matched (other, then)

Pattern 3:

```
re.search(r'[^a-zA-Z][Tt]he[^a-zA-Z]',text)
```

Errors

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - **Increasing accuracy or precision** (minimizing false positives)
  - **Increasing coverage or recall** (minimizing false negatives).

Exercises

Regex Crossword

Define regular expressions to match strings that:

1. have a 't'
2. have a 't' or a 'T'
3. have a letter (and how many)
4. have a digit
5. have a decimal number
6. have a length higher than 3 characters
7. have an 'M' but not an 'm'
8. have a character repeated twice

Exercícios

9. Have only one character repeated many times
10. put all words between {}

Processamento de Linguagem Natural

Luís Filipe da Costa Cunha
lfilipecc1@gmail.com

José João Almeida
jj@di.uminho.pt