SICSS-Paris 2025
Regular Expressions

Étienne Ollion – CNRS | École polytechnique
(course initially co-taught w/ J. Boelaert)

Introduction

Sometimes, your data looks like this

\t\t\n Born on March, 26th 1924 \t\t\n

Or like this

<html>Title: “Election 2024”</html>

Introduction

What we want to be able to do

“\t\t\n Marie was Born on April, 15th 1992 in Paris \t\t\n”
“\t\t\n John was born on March, 12th (?) 1991 in N.Y.\t\t”

Who	Month	Day	Year	Place
Marie	April	15	1992	Paris
John	March	12	1991	N.Y.

Introduction

The classic response to this is “Regular Expressions”
also known as “rational expressions”, or “RegEx”

Introduction

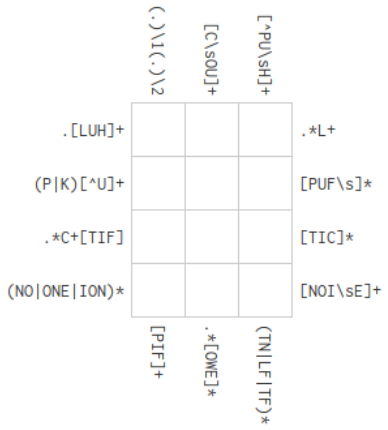
Regex: What are They?

- A powerful “Search and Replace” tool
  > Identify a pattern in the text, and do something to it

- They are central to working with unstructured data
  > Another tool for selection, on raw text this time

- A common tool among programmers

- An object of reverence and fascination



www.regexcrossword.com

Introduction

Regex: What are They?

And in Practice

- YAPL!
- But for our purpose, you will only learn a few basic principles
- Because for the rest, there is plenty of help online

Introduction

Outline

Regex: Basic Syntax

- Basic query
- Multiple choices
- Jokers
- Context
- Quantifiers
- Special characters

Regex: Basic Syntax

Regex: Basic Syntax
Regex serve to do basic searches
The search pattern is *exact*

Regex	Will match	Will not match
“a”	“Laura” “Alexia”	“Roberto” “Alexi”
“dataf”	“dataframe”	“data”
“19”	“1930”	“139”
“c_1”	“abc_1997”	“abc-1997”

Regex: Basic Syntax

Regex: Basic Syntax

Multiple choice

There are a few operators that will help you

- [az8] = a or z or 8
- [a-u] = All letters from a to u
- a|b = a or b

Regex	Yes	No
“[Pp]aul”	“Pauline” “epaule”	“Pau”
“P[a-z4]ul”	“Paula”, “Pbula” “P4ula”, “Poula”	“PAula”
“P[a-zA-Z0-9]ul”	“Pbul”, “P0ul”	“P*ul”
“Dupont|Dupond”	“Dupont”, “DuponDupond”	“Tintin”

Regex: Basic Syntax

Regex: Basic Syntax

Jokers

They help you out when you are unsure.

- \d = Any digit
- \w = Any word, or digit, or _
- . = Anything you want
- [^a1*] = Anything, except a, 1, or *

Regex: Basic Syntax

Regex: Basic Syntax

Context

They position your search pattern in the string

- ^ = indicates the beginning of a string
- $ = indicates the end of a string
- \b = indicates the boundary of a word (no sign after)

Regex	Yes	No
“^Paula”	“Paula likes John”	“John is liked by Paula”
“Paula$”	“John is liked by Paula”	“Paula likes John”
“Paul\b”	“Paul”	“Paula”

Regex: Basic Syntax

Regex: Basic Syntax

Quantifiers

How many of these patterns do you want?

- a{18} = a, 18 times in a row
- a{7,} = a, 7 or more times in a row
- a+ = a, once or more
- a* = a, zero times or more
- a? = 0 or 1 a (“is there an a?”)

Regex	Yes	No
“Pa+ul”	“Paul”, “Paaaul”	“Pul”
“Pa*ul”	“Pul”, “Paaaul”	“Pa#ul”
“\d{4}”	“1976”, “12345”	“123”
“a\d{3,}a”	“a123a”, “a1234a”	“a12a”

Regex: Basic Syntax

Regex: Basic Syntax

Special characters

Sometimes, two languages mix in unpredictable ways

- + * \ [] () ?

All of these are part of our common language (your string of character) and the regex language.

==> To capture them, you need to escape them (add \)

Regex: Basic Syntax

Regex: Basic Syntax

Questions?

Regex: Basic Syntax

Regex: Basic Syntax

Question

What will this pattern capture?

.*

And what about this one?

\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\b

In Practice

Search in Python

import re
items = [“apple”, “banana”, “cherry”]

[element for element in items if re.match("*a*", elements)]

> Outputs a list

In Practice

Replace in Python

```
import re
re.sub("a", "X", "banana")
```

In Practice

Extract in Python

```
text = "ID: 12345"
match = re.search(r"(\d+)", text)

text = "ID1: 12345, ID2: 23456, ID3: 34567”
match = re.findall(r"(\d+)", text)
```

In Practice

Resources

Regex can be hard, but most of what you'll have to do it easy.

In addition to this, there are countless resources online

- http://regex101.com → Test your regex (and add another \ in R)
- https://www.rexegg.com/