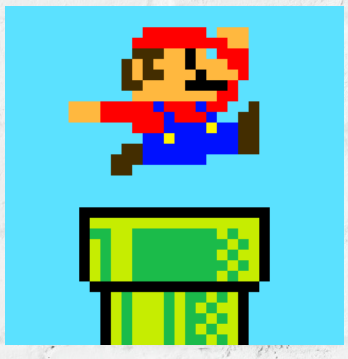
unix filters & regular expressions



RegEx

André Santos, afs@inesctec.pt

New students

Fill the form

`https://goo.gl/7DnCxC`

Repositories

- `github.com/andrefs/spln-docs`
- `github.com/andrefs/spln-2018-i`

Unix philosophy

- **Write programs that do one thing and do it well.**
- **Write programs to work together.**
- **Write programs to handle text streams, because that is a universal interface.**

Peter H. Salus, 1994

Unix filters

- software program that takes an input and produces an output
- can be used in a stream operation
- can be mixed and matched to create complex operations

Filter example: `tail`

- **input from STDIN or files (arguments)**

```
1  find | tail
2  tail my_large_file.txt
```

- **output to STDOUT (can be redirected)**

```
1  tail my_large_file | nl
2  tail my_large_file > last_lines.txt
```

- **behavior modified with command line arguments**

```
1  tail -n 20 my_large_file # show 20 lines
```

Unix filters

- **head**: first lines
- **tail**: last lines
- **cut**: remove columns
- **paste**: merge lines of files
- **nl**: number lines
- **wc**: count lines, words and chars
- **grep**: filter by pattern
- **cat**: (concat and) print

Unix filters

- **tr**: translate or delete characters
- **sed**: stream editor for filtering and transforming text
- **awk**: pattern scanning and processing language
- **perl**: Perl (can be used like awk)
- **sort**: sort lines of text files
- **uniq**: report or omit repeated lines
- **tee**: read from standard input and write to standard output and files

Filter composition

- **Calc. the most used bash commands**

```
history \
  | awk '{a[$4]++}END{for(i in a){print a[i] "\t" i}}' \
  | sort -rn \
  | head
```

- **Sort lines by the number of occurrences (desc)**

```
sort | uniq -c | sort -nr
```

- **Generate 100 random characters**

```
cat /dev/urandom \
  | tr -dc "0-9a-zA-Z!@#$%^&*_+-" \
  | head -c 100
```

System calls

```
1  import subprocess
2
3  exit_code = subprocess.call(["ls", "-l"])
4
5  # returns output as byte string
6  output = subprocess.check_output('ifconfig')
7
8  # using decode() function to convert byte string to string
9  print(returned_output.decode("utf-8"))
```

Unix filter with Python

```
1  import getopt
2  import sys
3
4  options, remainder = getopt.getopt(sys.argv[1:], 'abc:', \
5    ['output=', 'verbose', 'version='])
6  dict_opts = dict(options)
7
8  # input from STDIN or file
9  if remainder:
10   input = open(remainder[0])
11 else:
12   input = sys.stdin
13
14
15 # output to STDOUT or file
16 out = dict_opts.get('--output', None)
17 if out:
18   output = open(out, 'w+')
19 else:
20   output = sys.stdout
21
22 output.write("".join(input.readlines()))
```

Unix filter (simpler version)

```
1  import fileinput, getopt, sys, re
2  opts, args = getopt.getopt(sys.argv[1:], "d")
3
4  for line in fileinput.input(args):
5    line = line.strip()
6     # ...
```

Exercises

1. Write a sequence of Unix filters to print lines 40 to 50 of a file
2. Write a Python script which does the same thing by using system calls
3. Rewrite the script without using system calls

Regular expressions

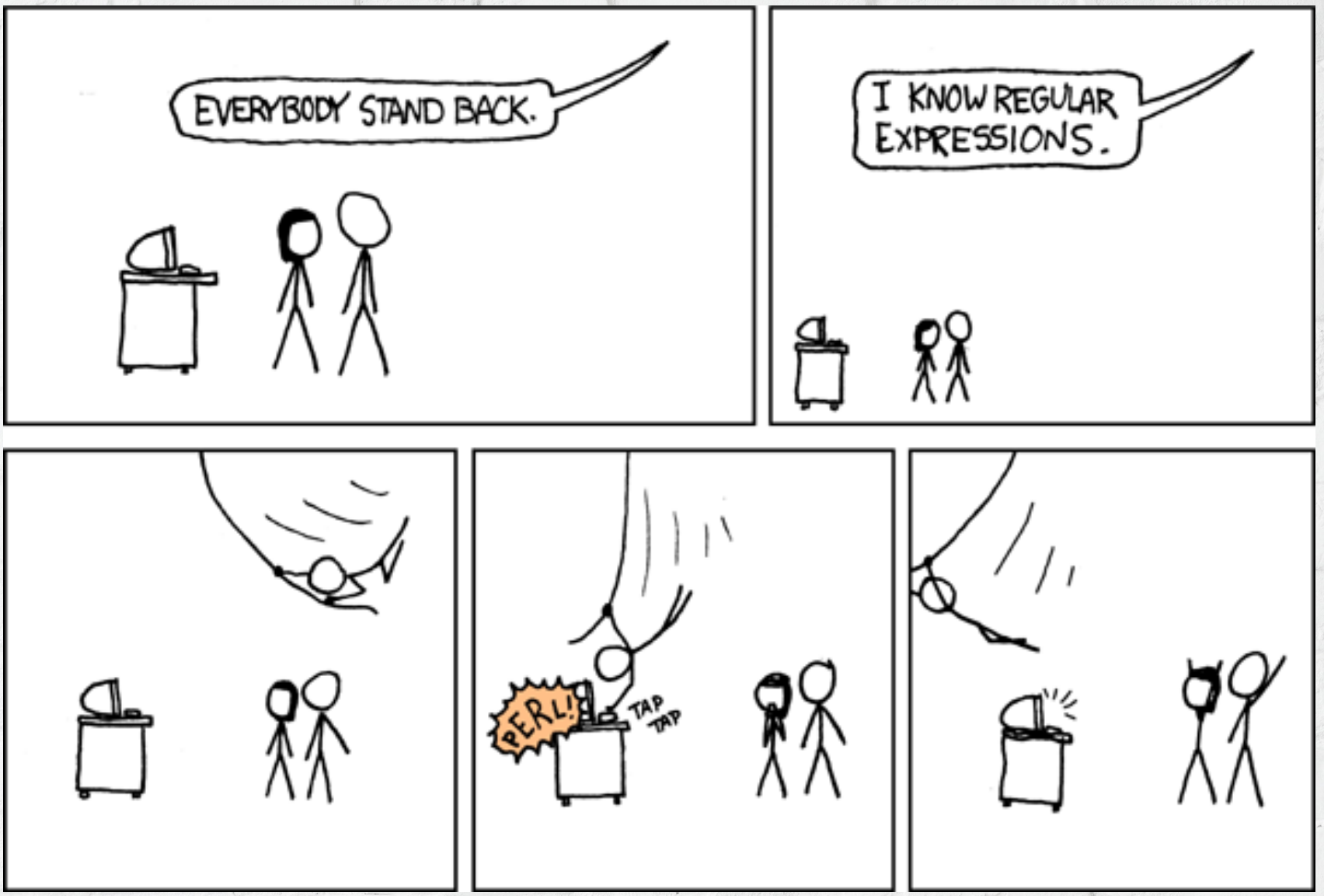
RegEx are a language for specifying text search strings.

- **They are used to specify patterns to be matched against strings.**
- **Tipically used in word processors and text editors, search engines and text processing utilities.**
- **Also, in programming languages:**
  - **Perl, Java `java.util.regex`, Python *re*, Ruby, ...**

Regular expressions

```
1  import re
2
3  string = 'This is my string'
4
5  re.findall(r'my', string)      # ['my']
6  re.sub(r'my', 'your', string) # 'This is your string'
```

Regular expressions



Background

- First originated in the 1950s
- Regular expressions can express regular languages (languages accepted by deterministic finite automata)
- Multiple standards, most common is POSIX

Basic

- r’amanhã’

```
1  re.search(r’amanhã’, ’O António chega amanhã.’)
```

- r’gat.’

```
1  re.search(r’sac.’, ’O João leva os livros numa saca.’)
```

Quantification

- r’golo’

```
1  re.search(r’golo’, ’Ronaldo chuta e... golo!!!’)
```

- r’goloo?’

```
1  re.search(r’goloo?’, ’Ronaldo chuta e... goloo!!!’)
```

Quantification

- r’golo+’

```
1  re.search(r’golo+’, ’Ronaldo chuta e... golooooooo!!!’)
```

- r’goloo*’

```
1  re.search(r’goloo*’, ’Ronaldo chuta e... golo!!!’)
```

- r’golo{2,5}’

```
1  re.search(r’golo{2,5}’, ’Ronaldo chuta e... golooo!!!’)
```

Grouping

- ’aldeão’

Grouping

- 'aldeão'
  - aldeãos, aldeões, aldeães

Grouping

- 'aldeão'
  - aldeãos, aldeões, aldeães

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

Disjunction and intervals

- `[AEIOU]`
- `[0123456789]`
- `alun[oa]`

Disjunction and intervals

- `[AEIOU]`
- `[0123456789]`
- `alun[oa]`
- `[A-Z]`
- `[0-9]`
- `[0-9A-F]`

Disjunction and intervals

- `[AEIOU]`
- `[0123456789]`
- `alun[oa]`
- `[A-Z]`
- `[0-9]`
- `[0-9A-F]`
- `[^aeiou]`

Character classes

- `\d` (digit)
  - `\D` (not `\d`)
- `\w` (letter, digit or underscore)
  - `\W` (not `\w`)
- `\s` (whitespace)
  - `\S` (not whitespace)

Anchors

- `^` **(begining of the line)**
- `$` **(end of the line)**
- `\b` **(word boundary)**

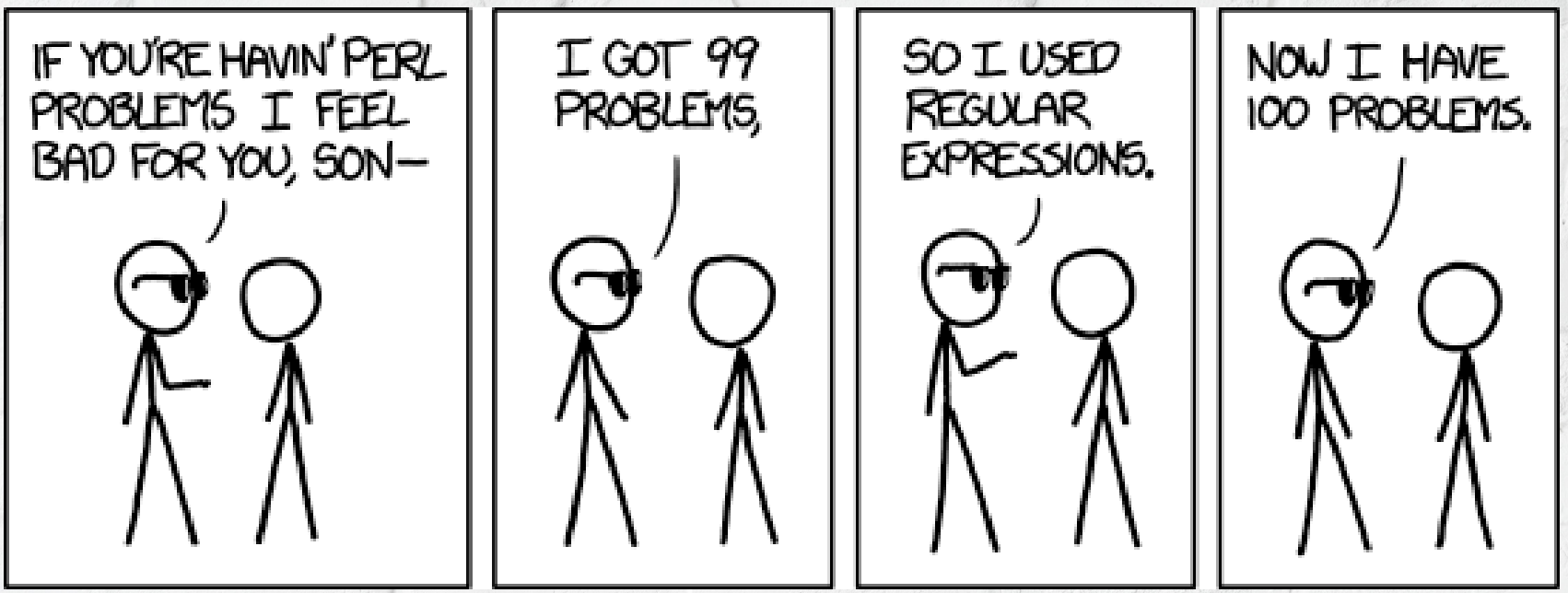
Capture groups

- `r'O ([A-Z][a-z]+) tem ([\w ]+)'`

```
1  re.findall(r'O ([A-Z][a-z]+) tem ([\w ]+)', \
2    'O Carlos tem uma mota.') # [('Carlos', 'uma mota')]
```

- `r'O ([A-Z][a-z]+) tem (?:[\w ]+)'`
  (don't capture the second group)

```
1  re.findall(r'O ([A-Z][a-z]+) tem (?:[\w ]+)', \
2    'O Carlos tem uma mota.') # ['Carlos']
```


IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.


I need a hand guys, does anyone have special skills in regular expressions?
Totally!
I'm your man!
Coming
Me! I fell in them when I was a kid
Ha ha, so funny...
Okay, still not matching...
Try it with ^[0-9] here?
Why don't you use #, wouldn't it be better?
What's that syntax coloring? It's terrible you can't read a thing
Not a chance it'll work if you don't escape the dot...
Your pattern sucks, I found an easier one here
Hmm... Still nothing...
CommitStrip.com

Example: URLs 

Example: URLs 

In most cases, you should not implement your own URL regex parser. Use 3rd party libraries such as Python's `urlparse`.

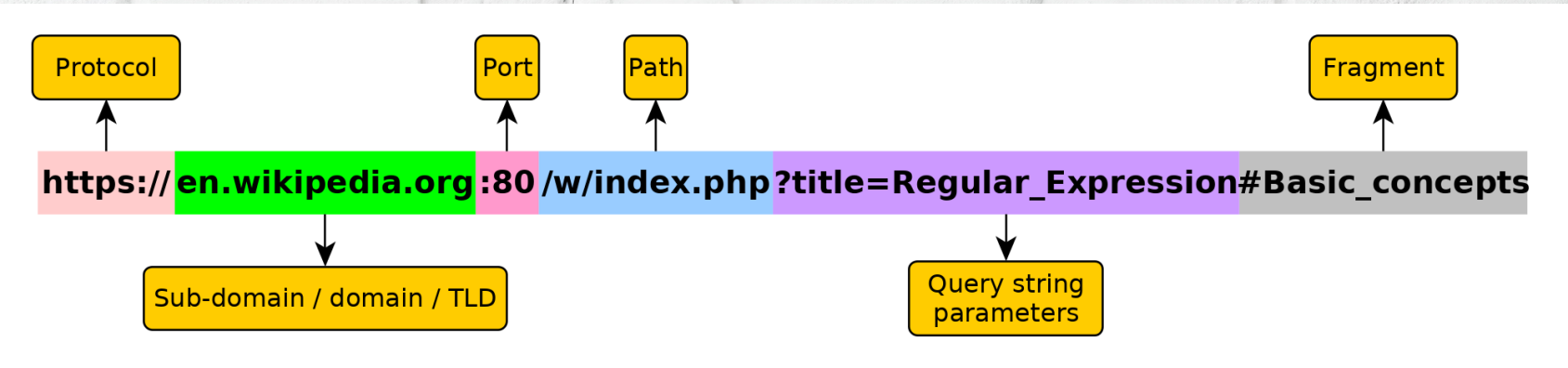
Example: URLs 

In most cases, you should not implement your own URL regex parser. Use 3rd party libraries such as Python's `urlparse`.

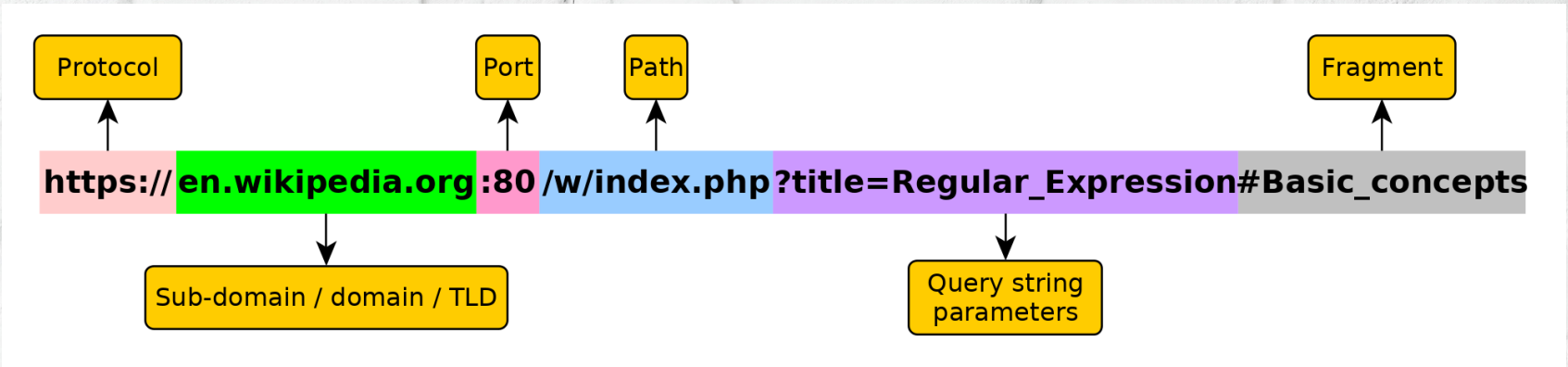
However, it always depends on what your use case is:

- **finding potential URLs in a text**
- **ensuring that a given URL is correctly formed**
- **...**

Example: URLs ⚠



Example: URLs 



Protocol: `http(s?)://`

Domain name: `[\w-]+(\.[\w-]+)*`

Port: `:\d{2,}`

Path: `/[\w-\.]+(/[\w-]+)*`

Query parameters: `/[\w-]+(/[\w-]+)*`

Fragment: `#[\w-]+`

Example: URLs ⚠

```
1  import re
2
3  protocol           = r'http(s?)://'
4  domain_name        = r'[\w-]+(\.[\w-]+)*'
5  port               = r':\d{2,}'
6  path               = r'/[\w\.-]+(/[\w\.-]+)*'
7  query_parameters   = r'\?[\w-]+=[\w-]+(&[\w-]+=[\w-]+)*'
8  fragment           = r'#[\w-]+'
9
10 url_re = protocol + domain_name + port + path + \
11   query_parameters + fragment
12
13 text = 'https://en.wikipedia.org:80/w/index.php' + \
14   '?title=Regular_expression#Basic_concepts'
15
16 re.search(url_re, text)
```

RegEx functions

- `re.search`
- `re.match`
- `re.findall`
- `re.sub`

Exercises II

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**

Exercises II

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

Exercises III

Create a file with some text (copy something from a news website, for example). Write Python scripts which receive text as input and:

1. **Outputs the text with marks on the begining of sentences (for example, '### ')**
2. **Finds and prints proper names**

Practical assignment #1

- Choose 1 of the 3 options (A, B or C)
- Groups of 2 or 3 elements
- Submission date: October 11, 2018
- Web address for submission will be announced in Blackboard

Practical assignment #1

- **A) Write a program to convert text to ASCII (remove accents). It should work as a Unix filter**
- **B) Given a file with a list of words (one word per line), find which words can be written as a sequence of chemical symbols (ex: "bacon" = Ba + Co + N)**

Practical assignment #1

- **c) Write a program which, given a large text with accents – "O João amanhã vai andar a pé (...)" – and a text with no accents – "O Ze tem um cao castanho (...)" – adds the accents to the second text.**

Practical assignment #1

Files and descriptions available at the SPLN 2018 repository.