
# REGEX

## Regular expressions and Scrapers

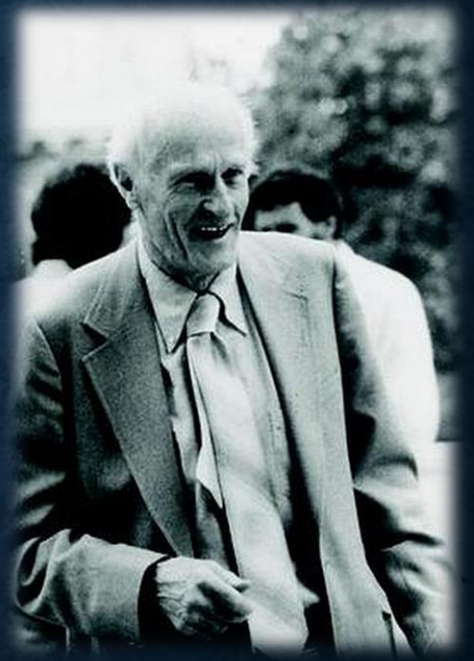
COS216
AVINASH SINGH
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF PRETORIA

# REGEX – OVERVIEW

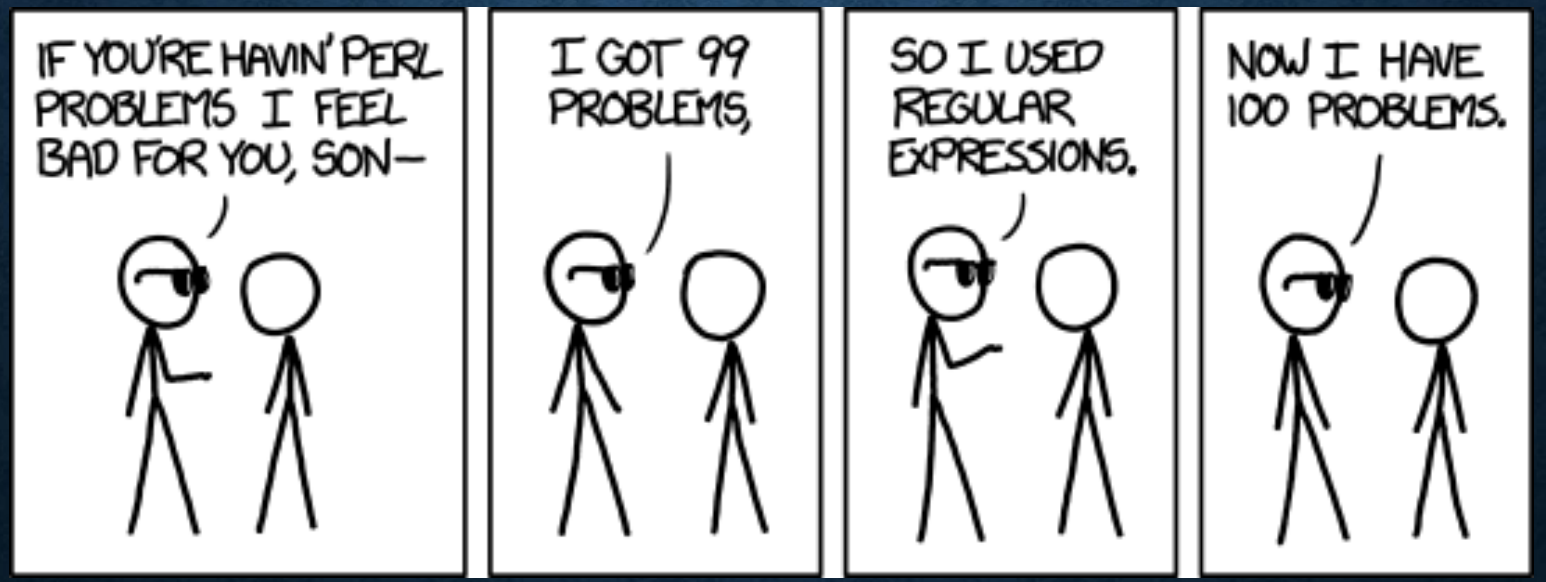
- Regular Expression (RegEx)
- Pattern notation used for
  - Finding/matching substrings (eg: check if birthdate was in the 90s)
  - Extracting substrings (eg: extract ZIP-code from address)
  - Finding and replacing substrings (eg: replace double spaces with a single space)
  - Validating strings (eg: validating email address format)

# REGEX – OVERVIEW

- In theoretical computer science and formal language theory, a sequence of characters that define a search pattern
- Original concept defined by mathematician Stephen Cole Kleene in the 1950s



# REGEX – OVERVIEW



# REGEX – OVERVIEW

- No RegEx standard exists
- Each language, library, and environment has its own flavour
  - However, they are still very similar
- Many variations exist
  - JavaScript
  - PHP
  - .NET
  - Python
  - POSIX
  - XRegExp
  - Many more …

# REGEX – OVERVIEW

- All RegEx operations could be done with basic string operations
  - indexOf: find the position of a substring
  - substring: extract a substring
  - replace: replace a substring

- However, basic string operations can become very cumbersome and requires a lot of code
- Rather use RegEx
  - Often a single line of code is required
  - Typically more efficient, due to better integrated search and backtracking algorithm

# REGEX – PROCESSOR

- A RegEx processor is a piece of software that can evaluate regular expressions
- A RegEx environment typically has the following
  - A haystack string to search in (to apply the pattern to)
  - A RegEx pattern to apply to the haystack string
    - In some environments the pattern is a simple string
    - In other environments RegEx is specifically accommodated, not requiring quotes
  - A test/match function returning a Boolean indicating if the pattern was found
  - A exec/search function returning a group of strings extracted from the haystack using the pattern

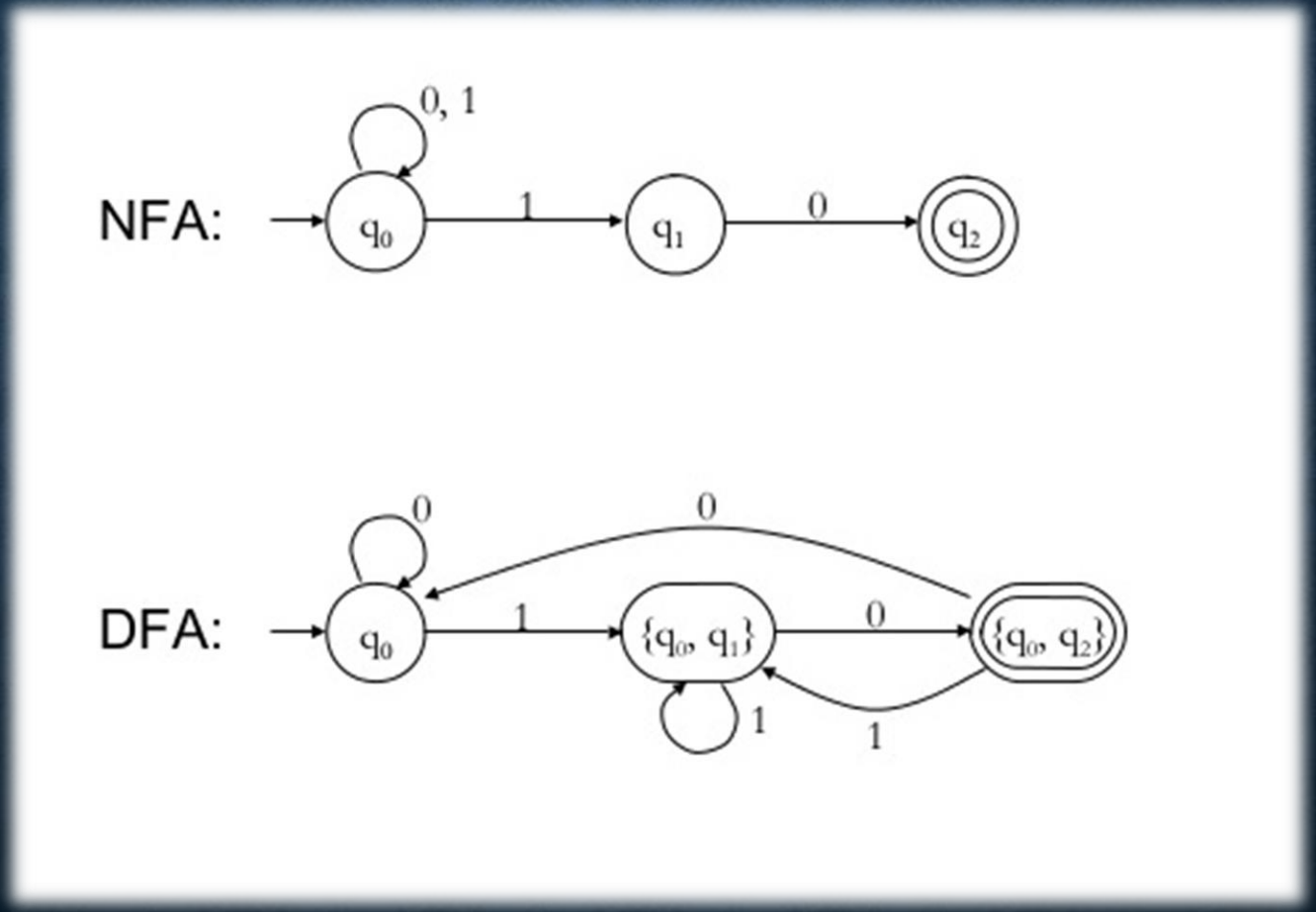
# REGEX – PROCESSOR

- A RegEx processor is a piece of software that can evaluate regular expressions
- A RegEx environment typically has the following
  - A haystack string to search in (to apply the pattern to)
  - A RegEx pattern to apply to the haystack string
    - In some environments the pattern is a simple string
    - In other environments RegEx is specifically accommodated, not requiring quotes
  - A test/match function returning a Boolean indicating if the pattern was found
  - A exec/search function returning a group of strings extracted from the haystack using the pattern

# REGEX – ENGINE

- RegEx can come with two different engines
- Both are finite machines and have the same capabilities and limitations
- Deterministic Finite Automatons (DFA)
  - Will always transition to a single state
  - Runs in linear time without backtracking
  - Each time goes through the same state transitions if the same input is fed
- Nondeterministic Finite Automatons (NFA)
  - Will transition to one or more states
  - Uses a greedy backtracking algorithm
  - Does not guarantee the same state transitions if the same input is fed

# REGEX – ENGINE



# REGEX – TOOLS

- Assemble and test your RegEx before integrating it into your code

https://regex101.com

https://www.debuggex.com

# REGEX – SYNTAX

- Grouping: ( )
- Or Operator: |
- And Operator: Does not exist, use consecutive grouping to accomplish this
- Not Operator: Does not exist, simply positively match the opposite

```
(BSc|BCom|BA) (Honors|Honours)
```

- Example returns strings that have BSc or BCom or BA, followed by Honors or Honours

# REGEX – QUANTIFICATION

| Pattern | Description |
| --- | --- |
| ? | 0 or 1 occurrence |
| * | 0 or more occurrences |
| + | 1 or more occurrences |
| {N} | Exactly N occurrences |
| {N,} | N or more occurrences |
| {N,M} | Between N and M occurrences |

# REGEX – POSIX

| Pattern | Description |
|---|---|
| . | Matches any character except newline, considered a literal dot inside [ ] |
| ^ | Matches the beginning of a line or string |
| $ | Matches the end of a line or string |
| \w | Matches alphanumeric characters (0-9, a-z, A-Z, _) |
| \W | Matches non-alphanumeric characters (equivalent to not \w) |
| \s | Matches whitespace characters (tab, newline, carriage return, space) |
| \S | Matches non-whitespace characters (equivalent to not \s) |
| \d | Matches digits (0-9) |
| \D | Matches non-digits (equivalent to not \d) |
| […] | Matches any of the individual characters between the brackets |

# REGEX – ESCAPING

- RegEx uses certain characters
- If you want to match these special characters, escape them
- Example
  - If you want to match the dollar sign $
  - $ is used by RegEx to match the end of a string
  - Use \$ to match the actual dollar sign

# REGEX – EXAMPLE

- Match a price of an item
- Decimal number with exactly 2 decimal places
- Example 256.02

# REGEX – EXAMPLE

- Equivalent to

```
\d+\.\d{2}
```

```
[0-9]+\.[0-9]{2}
```

- \d+: Match 1 or more digits
- \.: Match a dot
- \d{2}: Match exactly 2 digits for the decimal place

# REGEX – EXAMPLE

- Match University of Pretoria phone numbers
  - Pretoria's area code: 012
  - University of Pretoria's phone range: 420xxxx

- Match various formats
  - 0124201234
  - 012-420-1234
  - 012-4201234
  - 012 420 1234
  - 012 4201234

# REGEX – EXAMPLE

```
012[\s-]?420[\s-]?\d{4}
```

- 012: Exactly match the Pretoria area code
- [\s-]?: Match an optional white space or dash
- 420: Exactly match the UP range
- [\s-]?: Match an optional white space or dash
- \d{4}: Match any digit (0-9) exactly 4 times

# REGEX – EXAMPLE

- What is the difference between

```
012[\s-]?420[\s-]?\d{4}
```

- And

```
^012[\s-]?420[\s-]?\d{4}$
```

# REGEX – EXAMPLE

```
012[\s-]?420[\s-]?\d{4}
```

- Matches any substring within a larger string
- Will have a match for: 0124205263
- Will have a match for: asdsdfasdf01242052634234bg

```
^012[\s-]?420[\s-]?\d{4}$
```

- Matches the exact string from start to end, nothing can come before or after
- Will have a match for: 0124205263
- Will not have a match for: asdsdfasdf01242052634234bg

# REGEX – EXAMPLE

- Match an IPv4 address
- Each octet can only be between 0 and 255
- Example: 127.0.0.1

# REGEX – EXAMPLE

- Complex

```
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
```

- Or simplified

```
^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
```

# REGEX – EXAMPLE

- Match an email address
- Example: satoshi@gmail.com

# REGEX – EXAMPLE

- Match an email address

```
\w+@\w+\.\w+
```

# REGEX – EXAMPLE

- Previous RegEx does not allow subdomains: @cs.up.ac.za
- Accommodate multiple subdomains

```
\w+@\w+(\.\w+)+
```

# REGEX – EXAMPLE

- Previous RegEx does not allow symbols in name and domain
- Accommodate symbols

```
[\w._\-]+@[\w_\-]+(\.[\w_\-]+)+
```

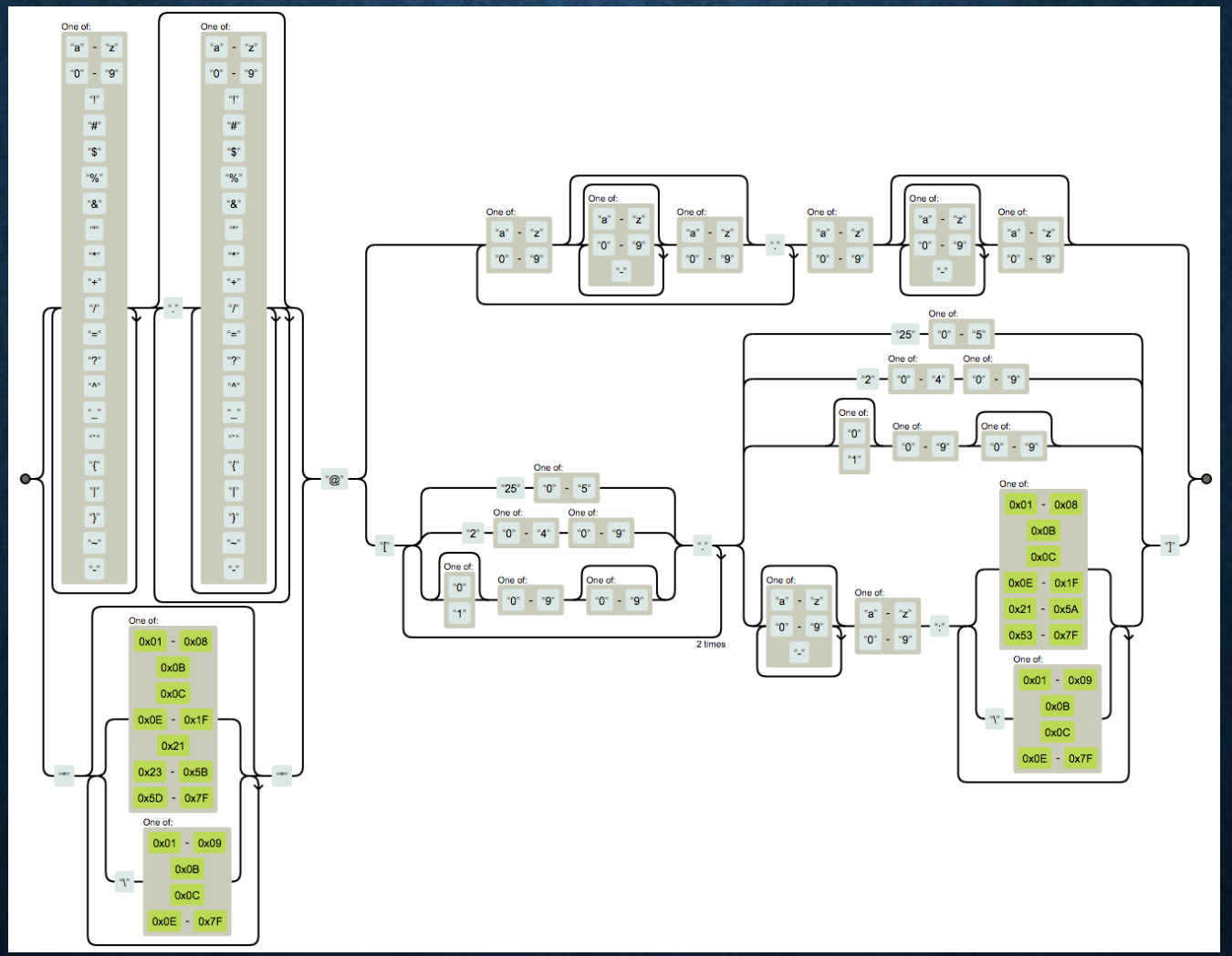
# REGEX – EXAMPLE

- Previous RegEx does still not accommodate all possible formats for email addresses
- There are restrictions on the domain characters
- Certain UTF-8 characters can be used
- Not all symbols can be used
- There is no perfect email RegEx

https://fightingforalostcause.net/content/misc/2006/compare-email-regex.php

https://www.regular-expressions.info/email.html

# REGEX – EXAMPLE



# REGEX – JAVASCRIPT

```
function validateEmail(email)
{
        var re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|
                ((".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|
                (([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
        return re.test(email);
}
```

# REGEX – PHP

```php
function validateEmail($email)
{
        $re = '/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|
                (".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|
                (([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/';
        return preg_match($re, $email) === 1;
}
```

# REGEX – PHP

- PHP has various integrate verification filters
- These filters are versatile and continuously maintained/updated
- Rather use one of these then creating your own RegEx

```php
function validateEmail($email)
{
        return filter_var($email, FILTER_VALIDATE_EMAIL);
}
```

# REGEX – APPLICATIONS

- Besides verification, RegEx is useful for parsing the output of other programs
- Example
  - You want to show your webserver's CPU and memory usage
  - Execute and external command in PHP ( eg: exec(…) )
  - On Linux you can use the top command/program ( eg: $output = exec("top") )
  - The program outputs a complex list of various values
  - Use RegEx to extract individual values (CPU and memory) from the output
  - Note that many cheap webservers have the exec function disabled
  - If you use a VPS or setup your own webserver, you have full access to all OS commands, so you can easily extract any information and even control the entire OS via a webinterface

# REGEX – SCRAPERS

- Another common use of RegEx is in scrapers
- Webcrawlers used by search engines are special scrapers
- Most websites do not have an easy-to-use API
  - However, the information is still displayed on their website
  - Make an AJAX request to the website to get the HTML
  - Use the DOM to navigate through the HTML and look for specific HTML elements
  - Retrieve the text from the elements and use RegEx to extract specific values from the text
  - Hence, even if a website does not have an API, you can write a small scraper to extract certain values from the website and then return it through your own custom API
  - Example, write a scraper that automatically searches through JunkMail, find cheap laptops, and then notify you if a good deal was found

# What is RegEx?

THIS: `str.match(/\d+\.\d+|\d+|[-+*/\(\)]/g);`

whaaaaat?????