Introduction to pattern matching

Using Regexes

CIRCL
Computer Incident
Response Center
Luxembourg

Alexandre Dulaunoy
alexandre.dulaunoy@circl.lu
Christian Studer
christian.studer@circl.lu

info@circl.lu

M2 SSI - 2024/01/26

Typical Linux problem

```
$ cat files.txt
readme.md
document.pdf
image.png
music.mp3
video.mp4
manual.pdf
```

Objectives: List only PDF files

$ man grep

```
GREP(1)                  User Commands                  GREP(1)

NAME
       grep, egrep, fgrep, rgrep - print lines that match
      patterns

SYNOPSIS
       grep [OPTION...] PATTERNS [FILE...]
       grep [OPTION...] -e PATTERNS ... [FILE...]
       grep [OPTION...] -f PATTERN_FILE ... [FILE...]

DESCRIPTION
       grep searches for PATTERNS in each FILE.  PATTERNS
      is one or more patterns separated  by  newline
      characters,  and  grep prints each line that matches
       a pattern.  Typically PATTERNS should be quoted
      when grep is used in a shell command.
```

Using grep

```
1 $ cat files.txt | grep 'pdf'
2 document.pdf
3 manual.pdf
```

Easy! However...

Using grep

```
1 $ cat files.txt | grep 'pdf'
2 document.pdf
3 manual.pdf
```

Easy! However...

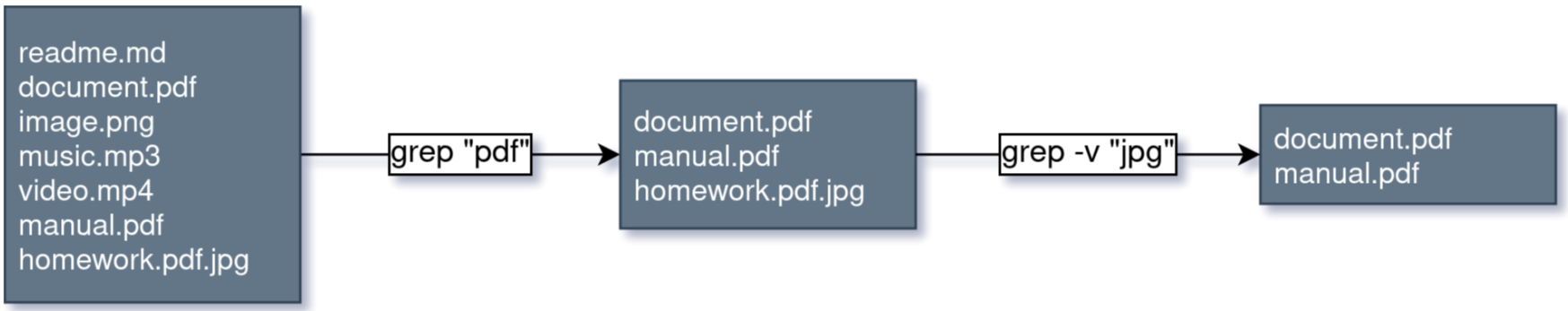
```
1 $ cat files-2.txt | grep 'pdf'
2 document.pdf
3 manual.pdf
4 homework.pdf.jpg
```

How can we filter out `homework.pdf.jpg`?

Using grep

```
$ cat files-2.txt | grep 'pdf' | grep -v 'jpg'
document.pdf
manual.pdf
```

-v allows us to perform an in**V**ert match



Using grep

```
$ cat files-3.txt | grep 'pdf' | grep -v 'jpg'
document.pdf
manual.pdf
adobe_pdf_reader.exe
i_hate_pdf.mp3
this.is.a.weird.pdf.filename.zip
filename with spaces are evil.pdf
```

Using invert match is not going to scale...

Other commonly encountered problems

- Matching valid email addresses
- Matching valid IBAN number
- Matching valid IPv4 addresses

$\rightarrow$ Regular Expressions (**Regex**) to the rescue

Regular Expression

A **regular expression** (shortened as **regex** or **regexp**) is a sequence of characters that specifies a search pattern in text.

Regexes are extremely useful in extracting information from text.

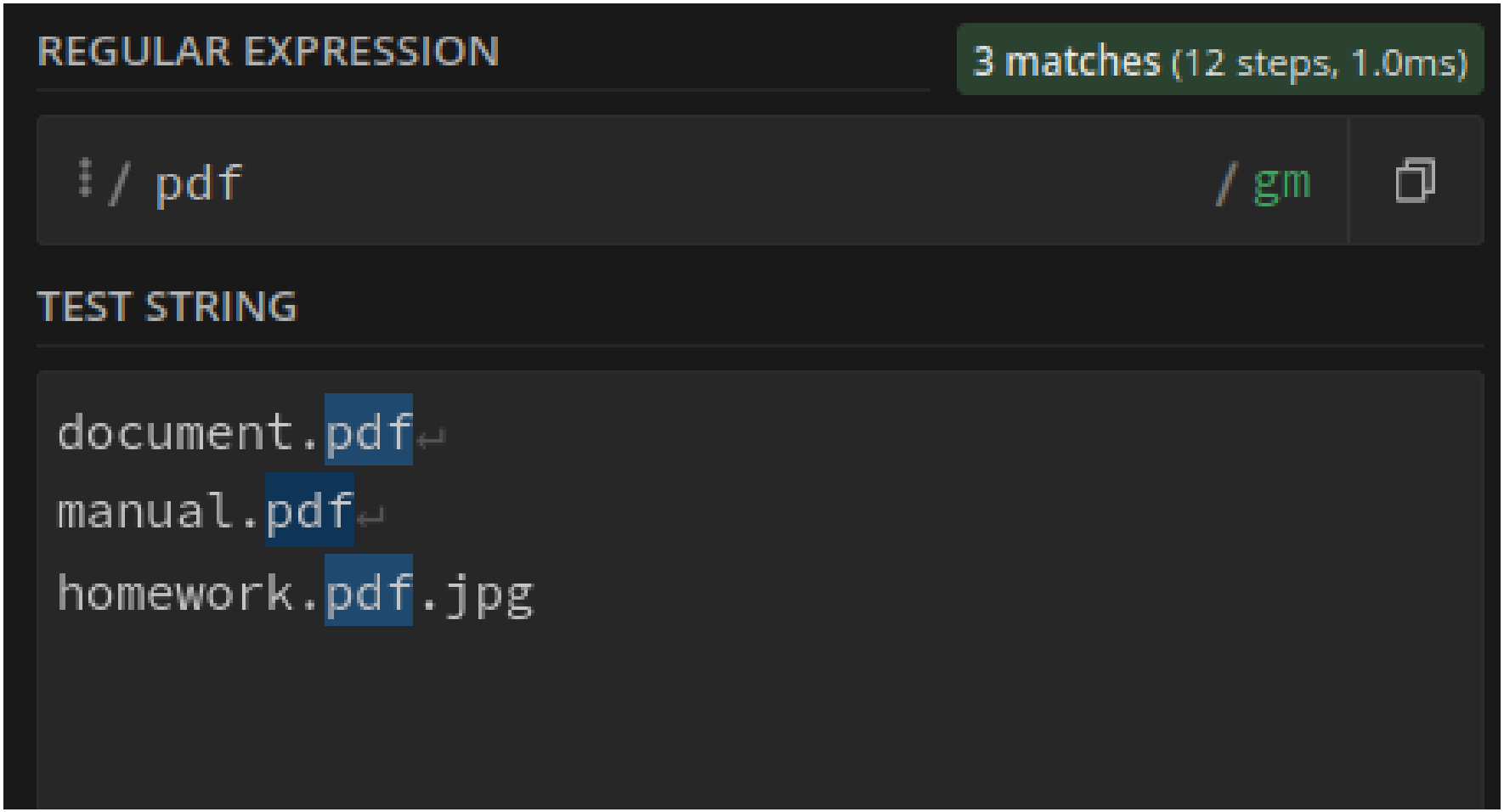
Regular Expression Basics

- What ?
  - Literal characters: `abc`
  - Quantifiers: `ab+c`
  - Operator OR: `(abc|cba)`
  - Bracket expressions: `[a-z]`
  - Meta sequences: `\S`
  - Capture group: `(abc)`
  - Anchors: `^abc$`
- New skill ?

```
/ <([a-z]+)(>(.*)<\/\1>|\s+\/>) /
```

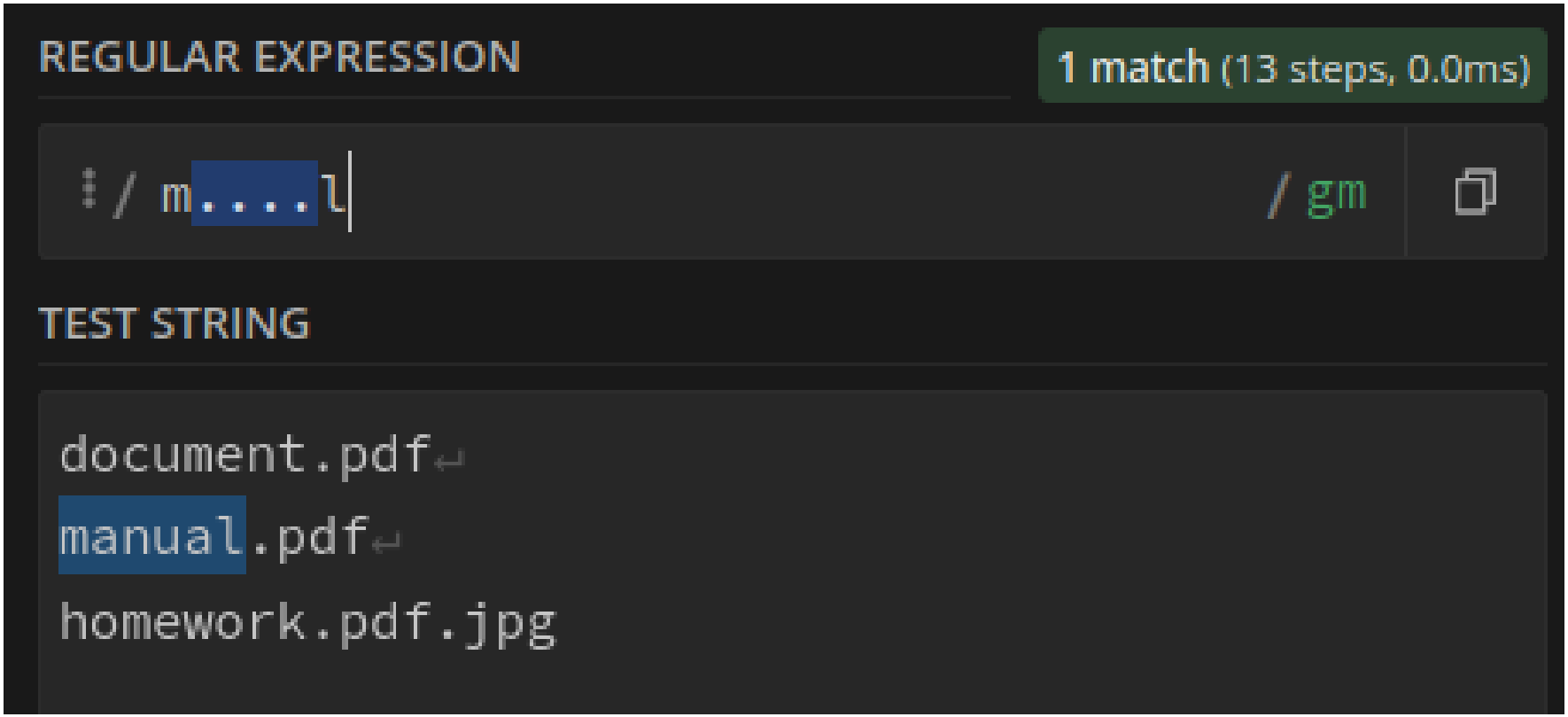
Regular Expression Basics: Literal characters

Letters and digits from the ASCII character set match their respective value



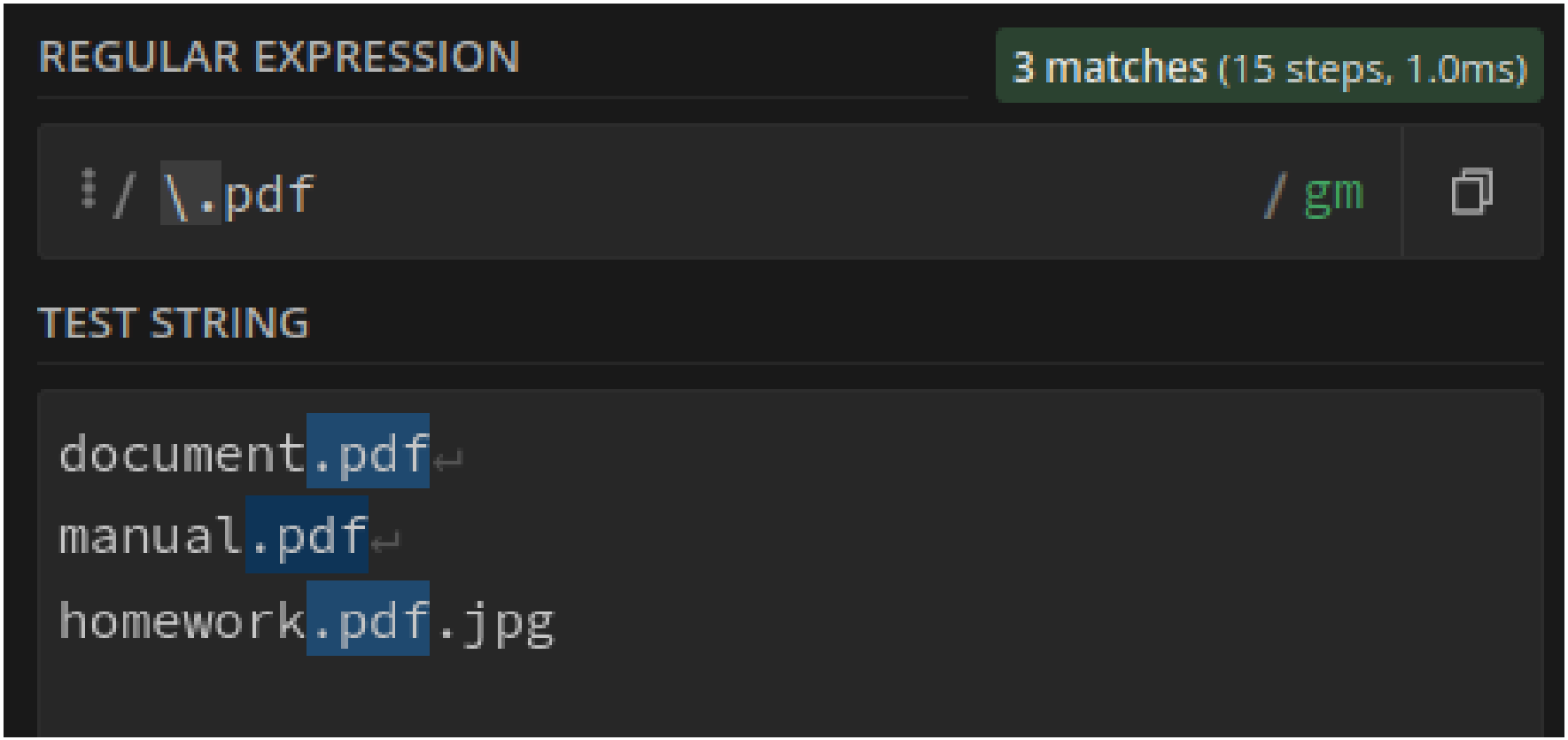
Regular Expression Basics: The Dot

- `.` is a *joker* or *wildcard* that can match any single character



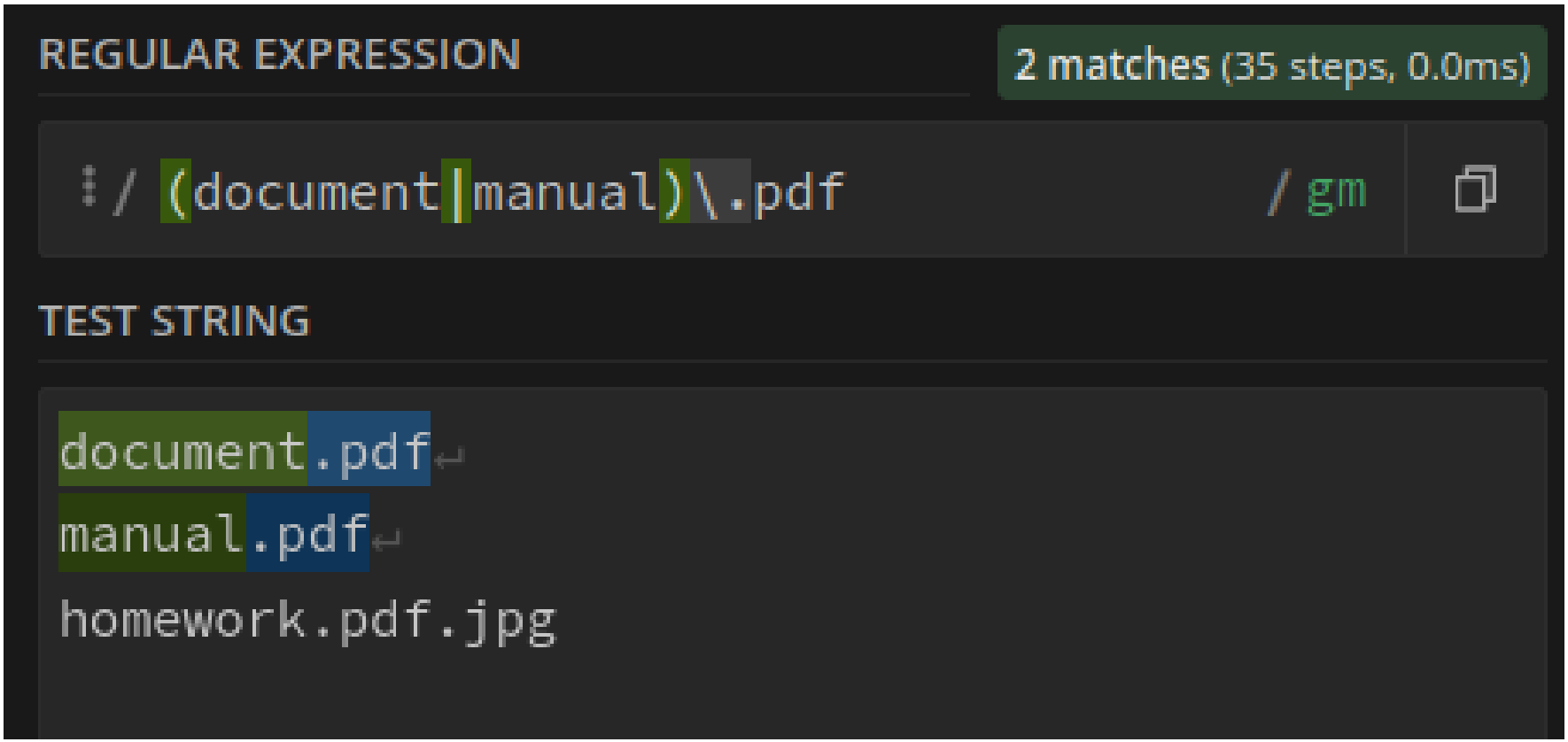
Regular Expression Basics: The Period

The period character `.` can be matched using the escape character `\.`



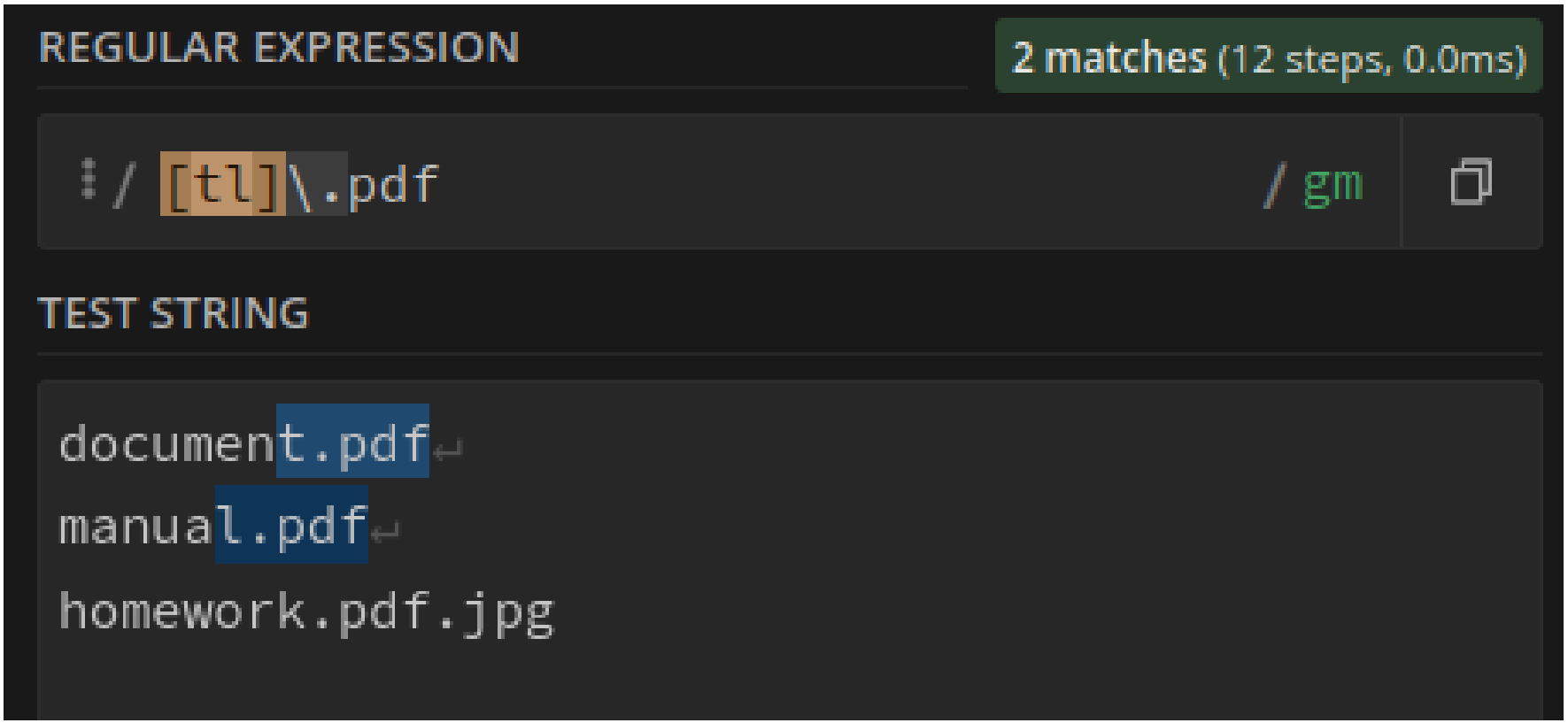
Regular Expression Basics: OR Operator

The (|) structure can be used as a logical operator to match one sequence or the other



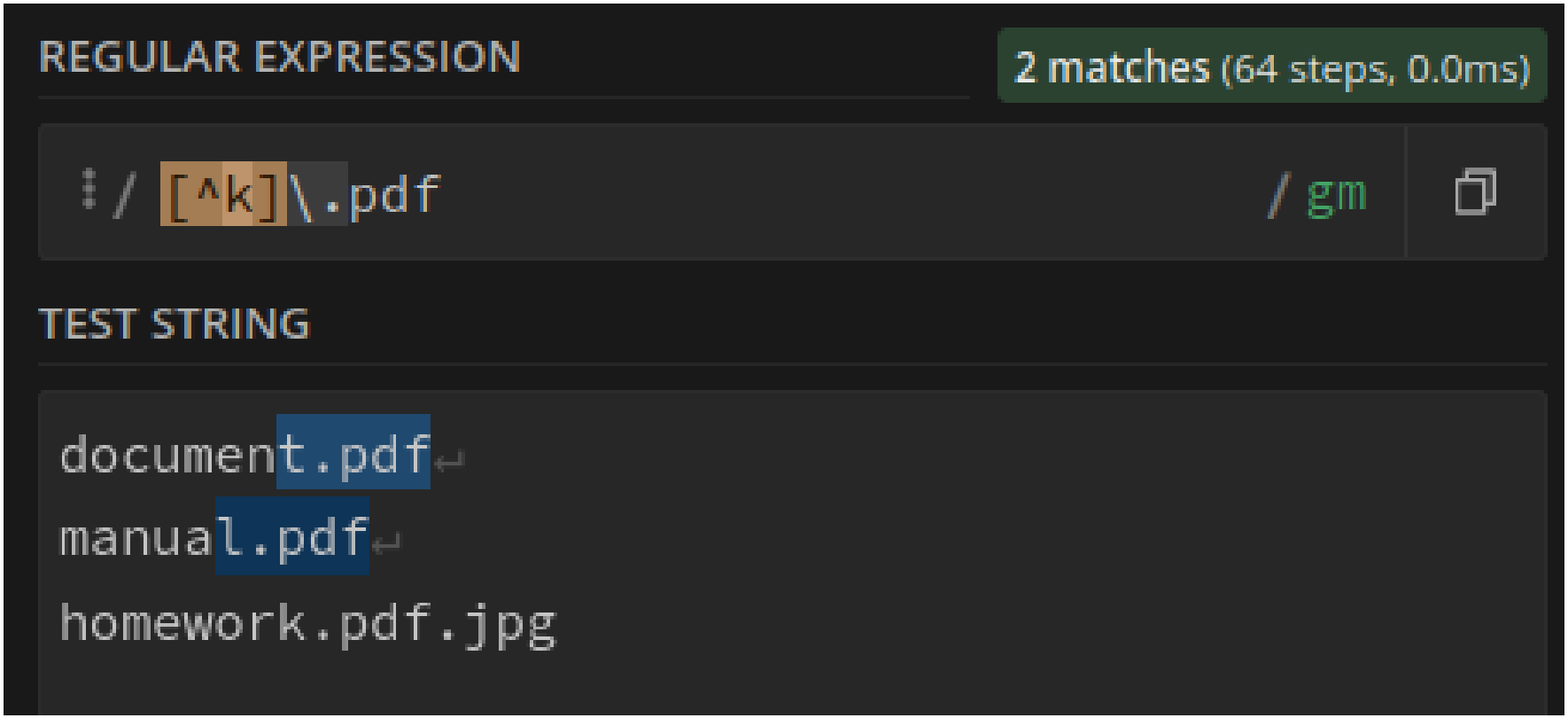
Regular Expression Basics: Bracket Expression (1)

The [ ] structure can be used to specifiy a set of characters that can match



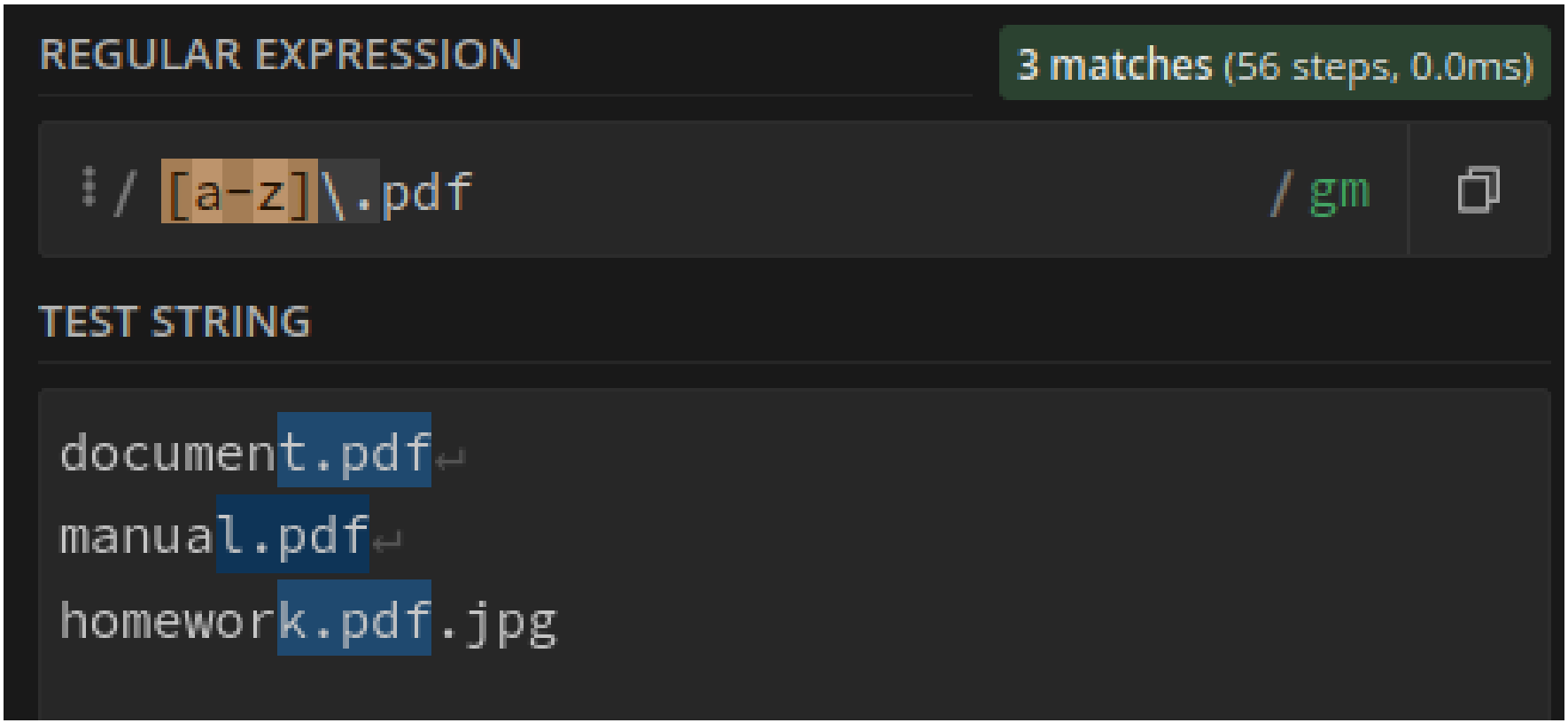
Regular Expression Basics: Bracket Expression (2)

The `[^ ]` structure can be used to exclude a specific set of characters



Regular Expression Basics: Bracket Expression (3)

The `[ - ]` structure can be used to specifiy a range of sequential characters



Regular Expression Basics: Meta Sequences (1)

- `/ . /`
  - Any single character
- `/ \w /`
  - Any word character
  - `/ [a-zA-Z0-9_] /`
  - **Match**: `any non-whitespace character $!-:;`
- `/ \W /`
  - Any non-word character
  - `/ [^a-zA-Z0-9_] /`
  - **Match**: `any whitespace character $!-:;`

Regular Expression Basics: Meta Sequences (2)

- `/ \d /`
  - Any digit
  - **Match**: 

- `/ \s /`
  - Any whitespace character
  - **Match**: 

- `/ \S /`
  - Any non-whitespace character
  - **Match**: 

Regular Expression Basics: Reading Exercises (1)

1. `/ facebo.k /`

Regular Expression Basics: Reading Exercises (1)

1. `/ facebo.k /`
   - **Match**: `facebook`, `faceboak`, `facebo&k`
2. `/ 4\.2 /`

Regular Expression Basics: Reading Exercises (1)

1. `/ facebo.k /`
   - **Match**: `facebook`, `faceboak`, `facebo&k`
2. `/ 4\.2 /`
   - **Match**: `4.2`
   - **Match**: `Nice number:  4.2`
3. `/ drink (beer|wine) ! /`

Regular Expression Basics: Reading Exercises (1)

1. `/ facebo.k /`
   - **Match**: `facebook`, `faceboak`, `facebo&k`
2. `/ 4\.2 /`
   - **Match**: `4.2`
   - **Match**: `Nice number:  4.2`
3. `/ drink (beer|wine) ! /`
   - **Match**: `I drink beer !`
   - **Match**: `I drink wine !`
4. `/ [e-h] /`

Regular Expression Basics: Reading Exercises (1)

1. `/ facebo.k /`
   - **Match**: `facebook`, `faceboak`, `facebo&k`
2. `/ 4\.2 /`
   - **Match**: `4.2`
   - **Match**: `Nice number:  4.2`
3. `/ drink (beer|wine) ! /`
   - **Match**: `I drink beer !`
   - **Match**: `I drink wine !`
4. `/ [e-h] /`
   - **Match**: `fefe`, `hehe`
   - **No match**: ~~`haha`~~

Regular Expression Basics: Writing Exercises (1)

1. **Match**: *red_light*, *green_light* and *!=_light*

Regular Expression Basics: Writing Exercises (1)

1. **Match**: *red_light*, *green_light* and *!=_light*

   `/ _light /`
2. **Match**: *red_light* and *green_light* **but not** *white_light*

Regular Expression Basics: Writing Exercises (1)

1. **Match**: *red_light*, *green_light* and *!=_light*

   `/ _light /`
2. **Match**: *red_light* and *green_light* **but not** *white_light*

   `/ (red|green)_light /`
3. **Match**: **_light* where `*` is any digit

Regular Expression Basics: Writing Exercises (1)

1. **Match**: *red_light*, *green_light* and *!=_light*

   `/ _light /`
2. **Match**: *red_light* and *green_light* **but not** *white_light*

   `/ (red|green)_light /`
3. **Match**: **_light* where `*` is any digit

   `/ [0-9]_light /`
4. **Match**: *?_light* where `?` is 4-letters color name

Regular Expression Basics: Writing Exercises (1)

1. **Match**: *red_light*, *green_light* and *!=_light*

   `/ _light /`
2. **Match**: *red_light* and *green_light* **but not** *white_light*

   `/ (red|green)_light /`
3. **Match**: **_light* where `*` is any digit

   `/ [0-9]_light /`
4. **Match**: *?_light* where `?` is 4-letters color name

   `/ [a-z][a-z][a-z][a-z]_light /`

Regular Expression Basics: Writing Exercises (1)

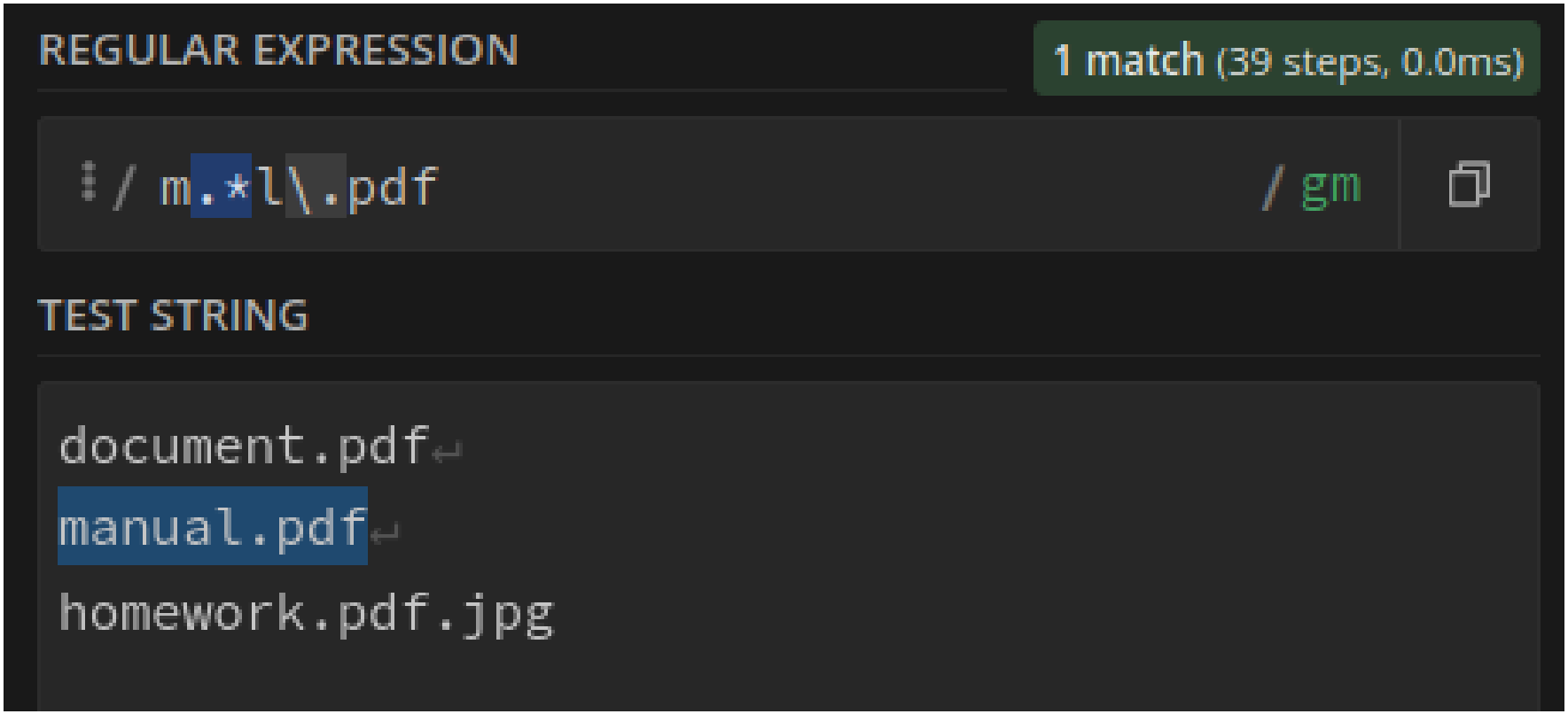
1. **Match**: *red_light*, *green_light* and *!=_light*
   `/ _light /`
2. **Match**: *red_light* and *green_light* **but not** *white_light*
   `/ (red|green)_light /`
3. **Match**: **_light* where `*` is any digit
   `/ [0-9]_light /`
4. **Match**: *?_light* where `?` is 4-letters color name
   `/ [a-z][a-z][a-z][a-z]_light /`

Question: *?_light* where `?` is any color between 3 and 6 letters

We need a way to express occurences... Introducing **quantifiers**

Regular Expression Basics: Quantifiers (1)

The * control character can be used to describe **zero or more** occurences



Regular Expression Basics: Quantifiers (2)

The + control character can be used to describe **one or more** occurences

Regular Expression Basics: Quantifiers (3)

- `/ a? /`
  - Match 0 or one a character
- `/ a{3} /`
  - Match exactly 3 a character
- `/ a{3,} /`
  - Match 3 or more a character
- `/ a{3,6} /`
  - Match between 3 and 6 a character

Regular Expression Basics: Reading Exercises (2)

1. `/ colou?r /`

Regular Expression Basics: Reading Exercises (2)

1. `/ colou?r /`
   - **Match**: `colour` and `color`
2. `/ go*gle /`

Regular Expression Basics: Reading Exercises (2)

1. `/ colou?r /`
   - **Match**: `colour` and `color`
2. `/ go*gle /`
   - **Match**: `gogle`, `gooooogle`, `ggle`, ...
3. `/ waz+up /`

Regular Expression Basics: Reading Exercises (2)

1. `/ colou?r /`
   - **Match**: `colour` and `color`
2. `/ go*gle /`
   - **Match**: `gogle`, `gooooogle`, `ggle`, ...
3. `/ waz+up /`
   - **Match**: `wazup`, `wazzzzzup`, ...
4. `/ +352[0-9]{6,8} /`

Regular Expression Basics: Reading Exercises (2)

1. `/ colou?r /`
   - **Match**: `colour` and `color`
2. `/ go*gle /`
   - **Match**: `gogle`, `gooooogle`, `ggle`, ...
3. `/ waz+up /`
   - **Match**: `wazup`, `wazzzzzup`, ...
4. `/ +352[0-9]{6,8} /`
   - **Match**: `+352791648`, `+35226791349`

Regular Expression Basics: Writing Exercises (2)

1. **Match**: The time (16:42, 03:59)

Regular Expression Basics: Writing Exercises (2)

1. **Match**: The time (16:42, 03:59)

   `/ [0-9][0-9]:[0-9][0-9] /`

   (not perfect but good enough for the exercise)
2. **Match**: Luxembourg postal code (L-4253, L-1110)

Regular Expression Basics: Writing Exercises (2)

1. **Match**: The time (16:42, 03:59)

   `/ [0-9][0-9]:[0-9][0-9] /`

   (not perfect but good enough for the exercise)
2. **Match**: Luxembourg postal code (L-4253, L-1110)

   `/ L-[0-9]{4} /`
3. **Match**: **_light* where * is any color?

Regular Expression Basics: Writing Exercises (2)

1. **Match**: The time (16:42, 03:59)

   `/ [0-9][0-9]:[0-9][0-9] /`

   (not perfect but good enough for the exercise)
2. **Match**: Luxembourg postal code (L-4253, L-1110)

   `/ L-[0-9]{4} /`
3. **Match**: **_light* where * is any color?

   `/ [a-z]+_light /`
4. **Match**: any hexadecimnal color (#ff0000, #f7f8f9)

Regular Expression Basics: Writing Exercises (2)

1. **Match**: The time (16:42, 03:59)

   `/ [0-9][0-9]:[0-9][0-9] /`

   (not perfect but good enough for the exercise)
2. **Match**: Luxembourg postal code (L-4253, L-1110)

   `/ L-[0-9]{4} /`
3. **Match**: **_light* where `*` is any color?

   `/ [a-z]+_light /`
4. **Match**: any hexadecimnal color (#ff0000, #f7f8f9)

   `/ #[a-f0-9]{6} /`

Regular Expressions: Final question

What does these regexes do?

1. `/ ([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01])) /`

2. `/ <([a-z]+)(>(.*)<\/\1>|\s+\/>) /`
   - `\1` is used to reference the first capturing group
   - First capturing group is `([a-z]+)`

Regexes: Going further

- ^ and $ anchors
- Capture **groups**
- **Greedy** and **Lazy** quantifiers
- **Possessive** quantifier

Linux commands and Regexes

Linux commands and Regexes

You have a small subset of a files crawled on Pastebin

```
$ du -hd1
158M    .
$ ls | wc
11875   11875  106875
```

Linux commands and Regexes

Let's list the number of files that have the `disney` keyword

```
$ grep -irl disney | wc
88       88      792
```

Let's filter on common email providers

```
$ egrep -ir 'disney'
    | egrep -i '(hotmail|live|outlook|gmail|yahoo)'
    | wc
52   122772 1566533
```

Linux commands and Regexes

Let's filter on the `email:password` pattern

```
$ egrep -ir 'disney'
    | egrep -i '(hotmail|live|outlook|gmail|yahoo)'
    | egrep -i '[a-zA-Z0-9_\-\+\.]+@\w+(\.\w+)+:\S+'
    | wc
13      902    21758
```

Let's extract all credentials and only the credentials

```
$ egrep -ir 'disney'
    | egrep -i '(hotmail|live|outlook|gmail|yahoo)'
    | egrep -io '[a-zA-Z0-9_\-\+\.]+@\w+(\.\w+)+:\S+'
    | wc
13       13      438
```

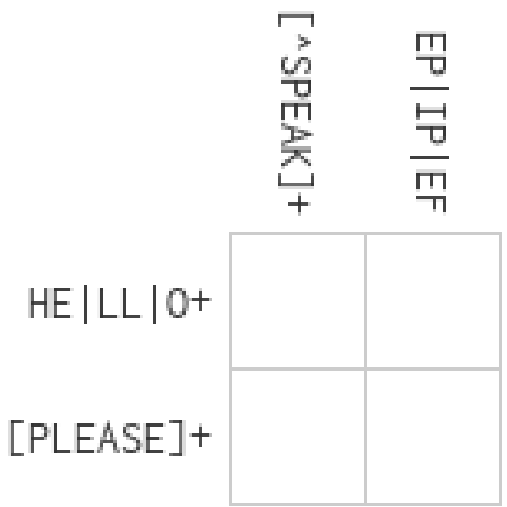
Linux commands and Regexes

Let's search for other type of accounts

```
$ egrep -ir 'psn'
    | egrep -i '(hotmail|live|outlook|gmail|yahoo)'
    | egrep -i '[a-zA-Z0-9_\-\+\.]+@\w+(\.\w+)+:\S+'
    | wc
2      124      583
```

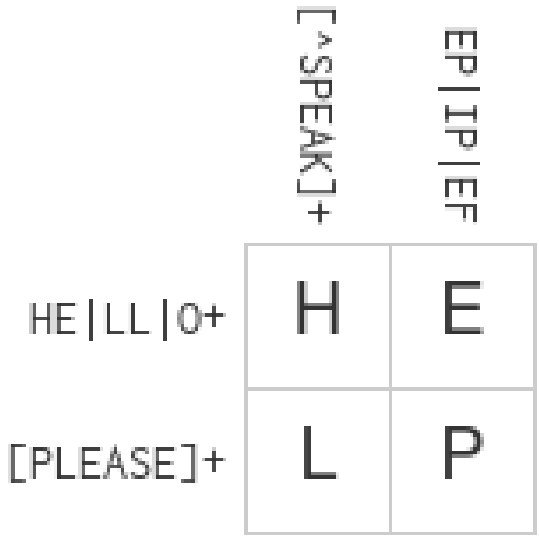
Fun with Regular Expressions

https://regexcrossword.com/



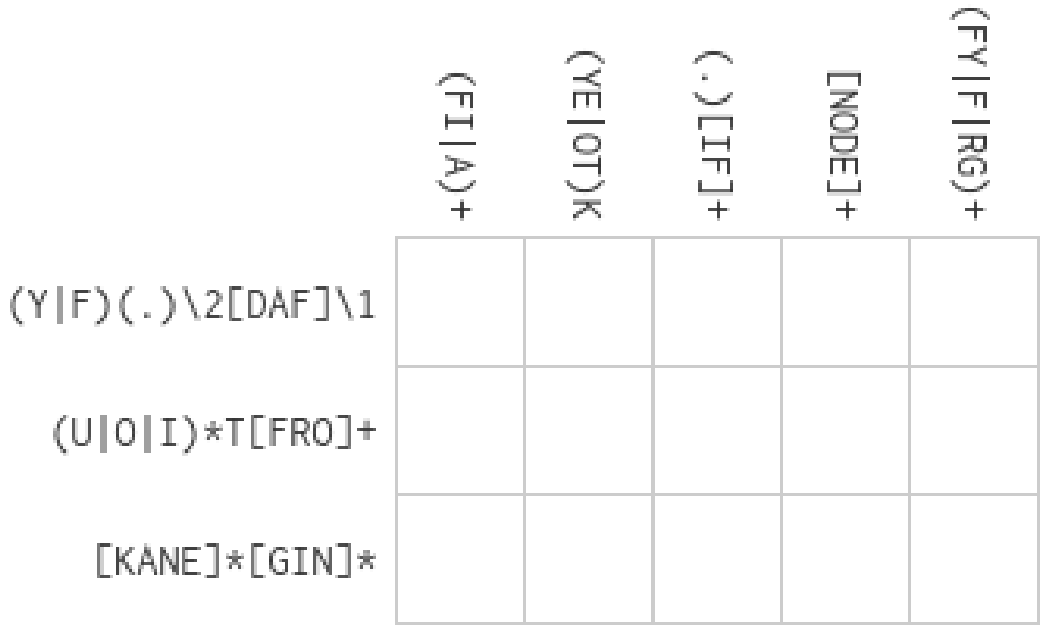
Fun with Regular Expressions

https://regexcrossword.com/



Fun with Regular Expressions

`https://regexcrossword.com/`



Fun with Regular Expressions

`https://regexcrossword.com/`

	`(FI|A)+`	`(YE|OT)K`	`(.)[IF]+`	`[NODE]+`	`(FY|F|RG)+`
`(Y|F)(.)\2[DAF]\1`	F	O	O	D	F
`(U|O|I)*T[FRO]+`	I	T	F	O	R
`[KANE]*[GIN]*`	A	K	I	N	G

Simple CTF challenge with regular expressions

- Hidden 4 flags in the dump
- They have the following pattern
  - `flag` letters in any order
  - `_` The underscore character
  - 2 upper case character
  - at least one special character from the list `@` , `&`, `:`
  - 2 lower case character
- Example: `falg_AA&:bb`

Simple CTF challenge with regular expressions

SHA1 Sum of the flags

```
1 $ echo 'falg_AA&:bb' | md5sum
2 ee9326ee21572fe4bba8e686a7ba6e5e
```

falg_AA&:bb	ee9326ee21572fe4bba8e686a7ba6e5e
?	53923b8f8490072107b3e8bb614749ce
?	429698c6d1742f02212ae89d3696577d
?	40178e8ef4264385fb7194176faf2318

Annex

Regular Expressions: Tag matching (1)

Create a Regex that validates the following tags:

```
namespace:predicate
namespace:predicate="value"
name_space:pred_icate="value"
namespace:predicate="qwert _+$- yuiop"
namespace:predicate="qwert=:yuiop"
```

But not these:

```
tag
name space:pred icate="value"
name-space:predicate="value"
namespace:predicate="qwert"yuiop"
```

Regular Expressions: Tag matching (2)

A valid tag is composed of 2 or 3 parts

```
namespace:predicate
namespace:predicate="value"
```

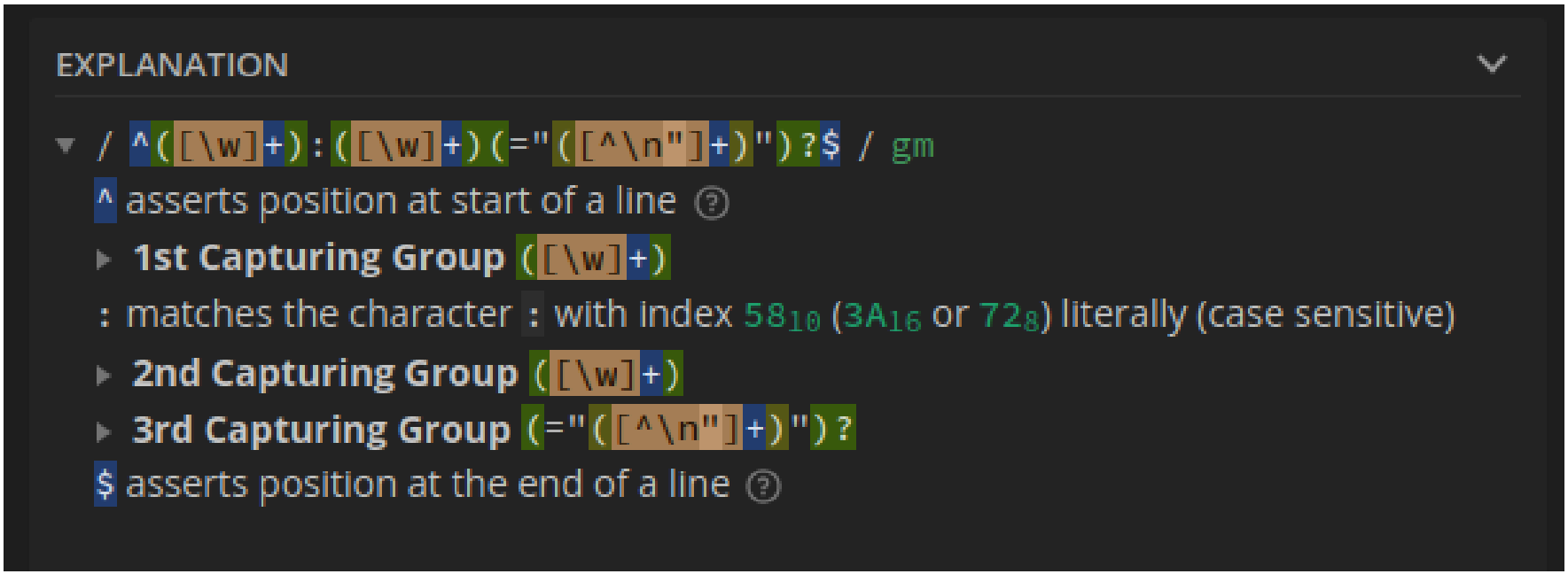
1. The `namespace`
2. The `predicate`
3. The optional `value`

Regular Expressions: Tag matching (3)

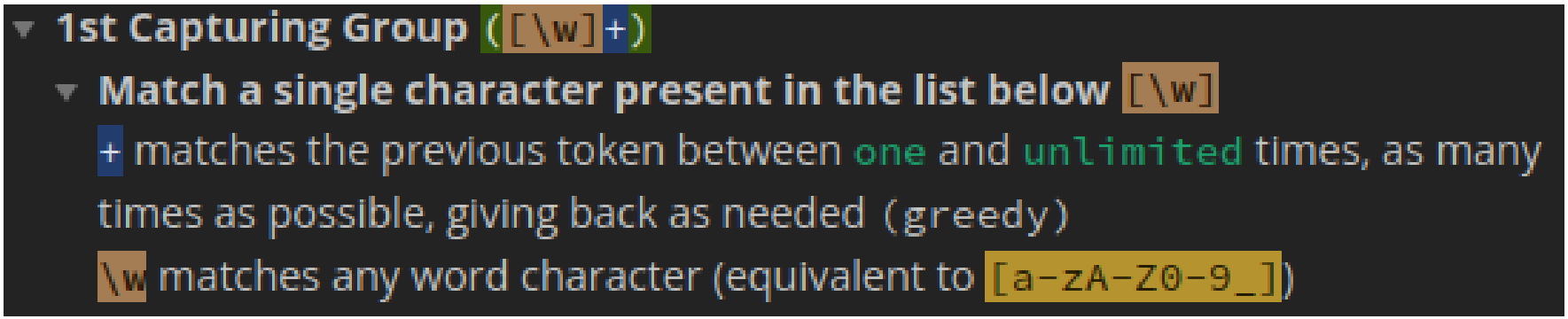
Tag validator

```
/ ^([\w]+):([\w]+)(="([^\n"]+)")?$ /
```

Regular Expressions: Tag matching (4)



Regular Expressions: Tag matching (5)



Regular Expressions: Tag matching (5)

- **3rd Capturing Group** `(="([^\n"]+)")?`
  - `?` matches the previous token between `zero` and `one` times, as many times as possible, giving back as needed `(greedy)`
  - `="` matches the characters `="` literally (case sensitive)
  - **4th Capturing Group** `([^\n"]+)`
  - `"` matches the character `"` with index 34_{10} (22_{16} or 42_{8}) literally (case sensitive)

- **4th Capturing Group** `([^\n"]+)`
  - **Match a single character not present in the list below** `[^\n"]`
    - `+` matches the previous token between `one` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`
    - `\n` matches a line-feed (newline) character `(ASCII 10)`
    - `"` matches the character `"` with index 34_{10} (22_{16} or 42_{8}) literally (case sensitive)