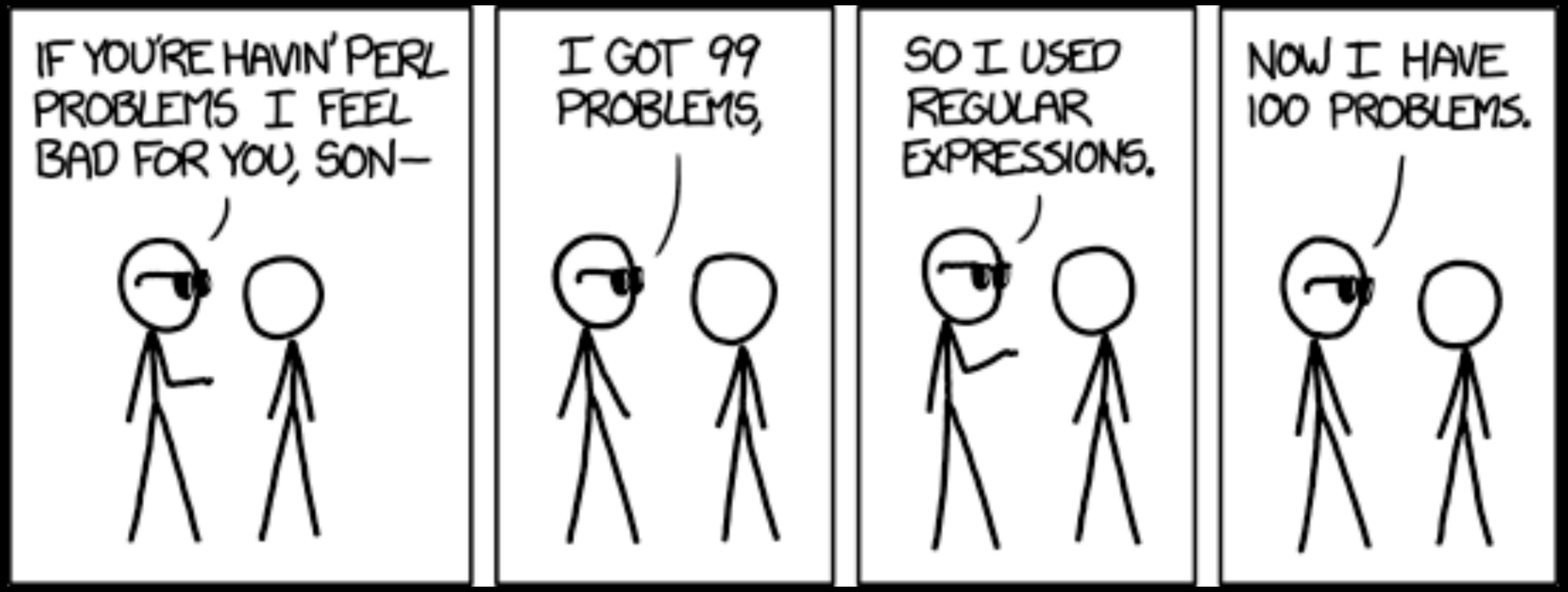
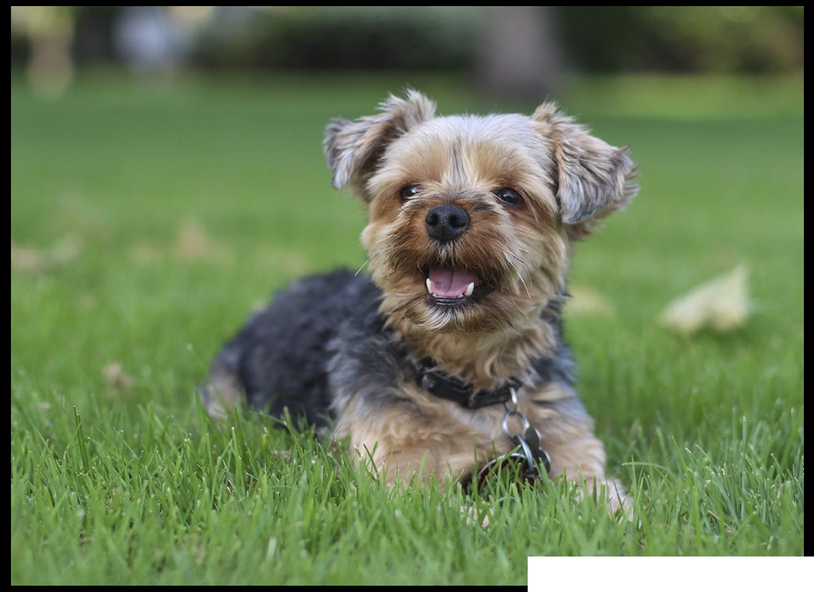
99 problems…
and regex

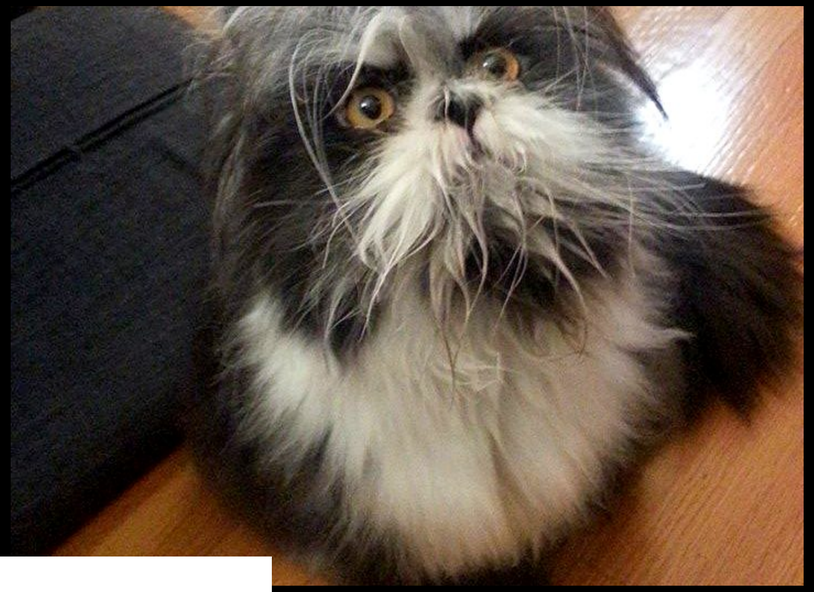
Mike Dowler

About me


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.

Regex, huh?
What is it good for?





Dog?

Scenario #1: Version numbers

How do we check if a software version meets or exceeds a given value?

`4.9.0.504`

`5` ✔

`4.9.9.9.9` ✔

`4.9.0.0.504` ✘

`4.9.0.503` ✘

`4` ✘

`4.10.0` ✔

Start simple

```
4.9.0.504
```

```
4\.9\.0\.504
```

Start simple

```
4.9.0.504
```

```
4\.9\.0\.504
```

```
3.14.9.0.504.7
```

Start simple

```
4.9.0.504
```

```
^4\.9\.0\.504$
```

Start simple

```
4.9.0.504
```

```
^4\.9\.0\.504
```

Build it up - one digit

```
4.9.0.504
```

Build it up - one digit

```
4.9.0.504
```

```
^4\.9\.0\.50[456789]
```

Build it up - one digit

```
4.9.0.504
```

```
^4\.9\.0\.50[456789]
```

```
^4\.9\.0\.50[4-9]
```

Build it up - two digits

```
4.9.0.504
```

Build it up - two digits

```
4.9.0.504
```

```
^4\.9\.0\.5[1-9][0-9]
```

Build it up - two digits

```
4.9.0.504
```

```
^4\.9\.0\.5[1-9][0-9]
```

```
^4\.9\.0\.5[1-9]\d
```

Build it up - one or two digits

```
^4\.9\.0\.50[4-9]
```

OR

```
^4\.9\.0\.5[1-9]\d
```

Build it up - one or two digits

```
^4\.9\.0\.50[4-9]
```

OR

```
^4\.9\.0\.5[1-9]\d
```

```
^4\.9\.0\.(50[4-9]|5[1-9]\d)
```

Build it up - three digits

4.9.0.504

Build it up - three digits

4.9.0.504

```
^4\.9\.0\.(50[4-9]|5[1-9]\d|[6-9]\d\d)
```

Build it up - three digits

4.9.0.504

```
^4\.9\.0\.(50[4-9]|5[1-9]\d|[6-9]\d\d)
```

```
^4\.9\.0\.(50[4-9]|5[1-9]\d|[6-9]\d{2})
```

Build it up - beyond three digits

```
4.9.0.504
```

```
^4\.9\.0\.(50[4-9]|5[1-9]\d|[6-9]\d{2}|
\d{4})
```

Build it up - beyond three digits

```
4.9.0.504
```

```
^4\.9\.0\.(50[4-9]|5[1-9]\d|[6-9]\d{2}|
\d{4})
```

```
^4\.9\.0\.(50[4-9]|5[1-9]\d|
[6-9]\d{2}|\d{4,})
```

Quantifiers

```
? = {0,1}
```

```
* = {0,}
```

```
+ = {1,}
```

Version numbers

4.9.0.504

```
^(4\.(9\.(0\.(50[4-9]|5[1-9]\d|
[6-9]\d{2}|\d{4})|[1-9])|[1-9]\d)|[5-9]|
[1-9]\d)
```

Scenario #2: Log file parsing

How can we extract (semi-)unpredictable data from predicatable contexts?

```
<backupConfig>
      <backupSets>
        <backupSet id="x">
          <name>yyyyy</name>
          …
        </backupSet>
        <backupSet id="x">
          <name>yyyyy</name>
          …
        </backupSet>
```

Find the string - part 1

```
^\s*<backupSet id="\d+">
```

```
\s matches space, tab or newline
```

```
\d matches any digit
```

Find the string - part 2

```
^\s*<backupSet id="\d+">\s*<name>.+<\/name>
```

`.` matches anything *except* newline

`/` needs to be escaped

Find the data in the string

Find the data in the string

```
^\s*<backupSet id="\d+">\s*<name>.+<\/name>
```

Find the data in the string

```
^\s*<backupSet id="\d+">\s*<name>.+<\/name>
```

```
^\s*<backupSet id="(\d+)">\s*<name>(.+)<\/name>
```

Find the data in the string

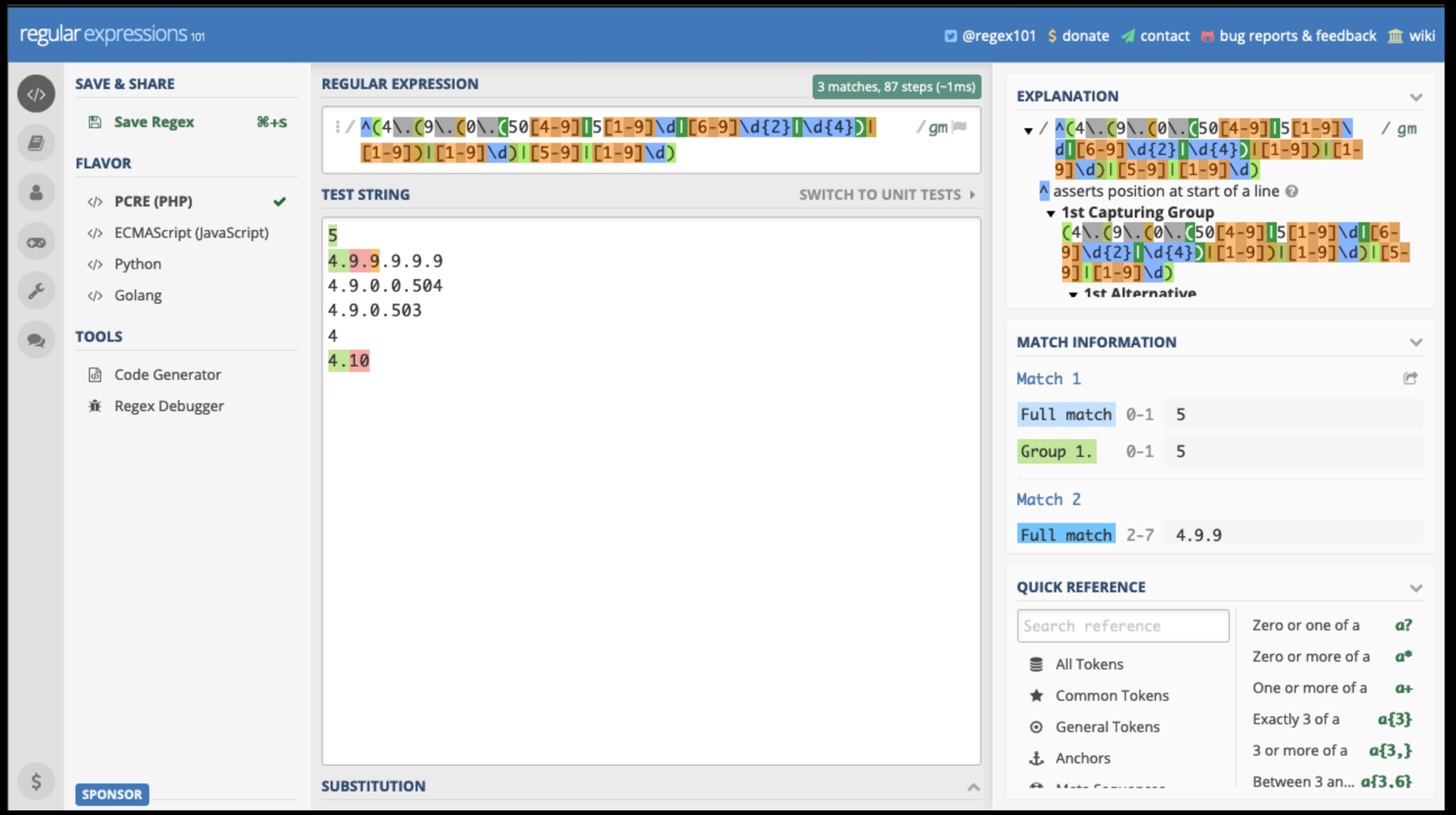
```
^\s*<backupSet id="\d+">\s*<name>.+<\/name>
```

```
^\s*<backupSet id="(\d+)">\s*<name>(.+)<\/name>
```

```
^\s*<backupSet id="(?P<id-cap>\d+)">\s*<name>(?
P<name-cap>.+)<\/name>
```

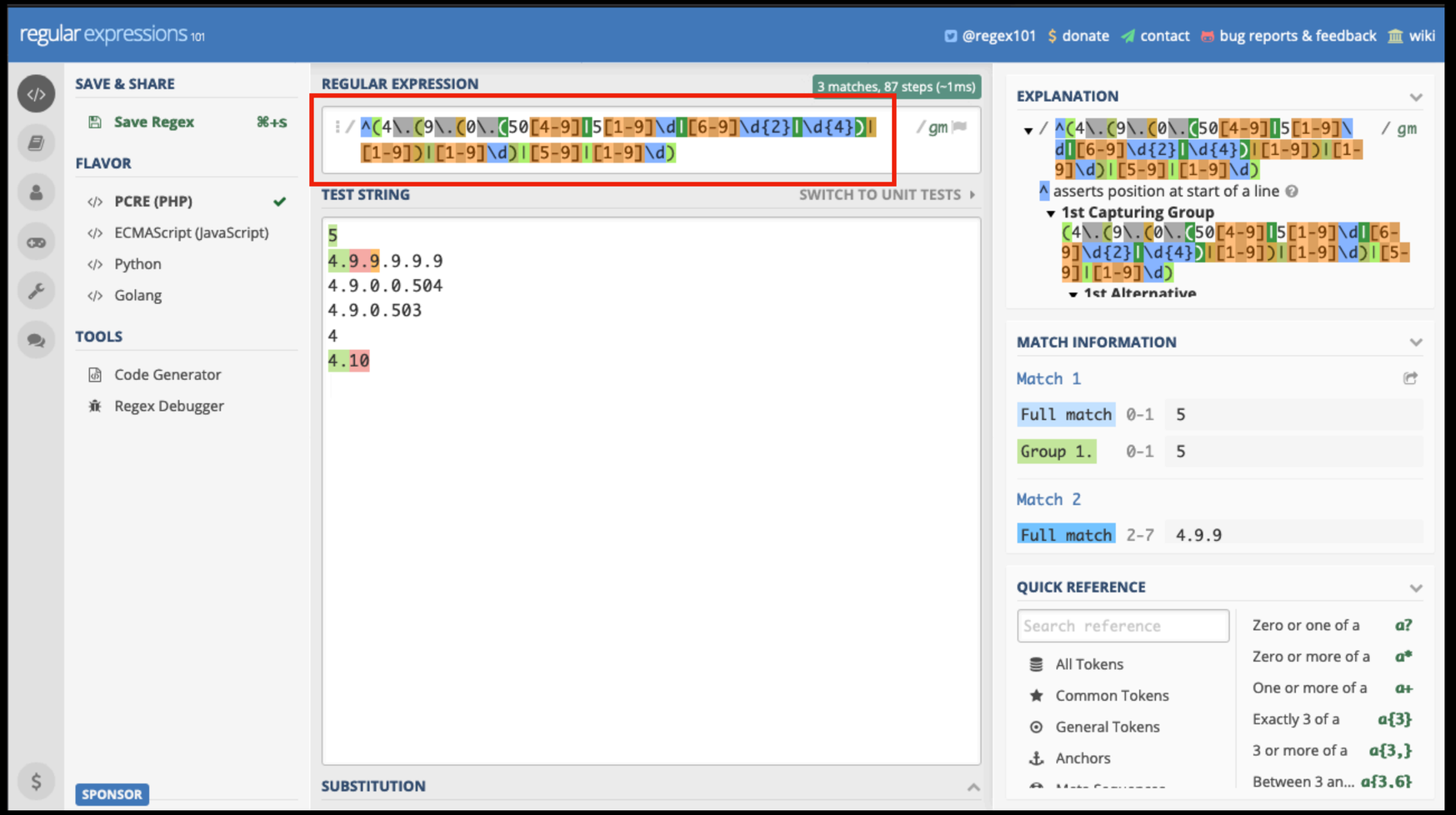
Practice!

https://regex101.com/



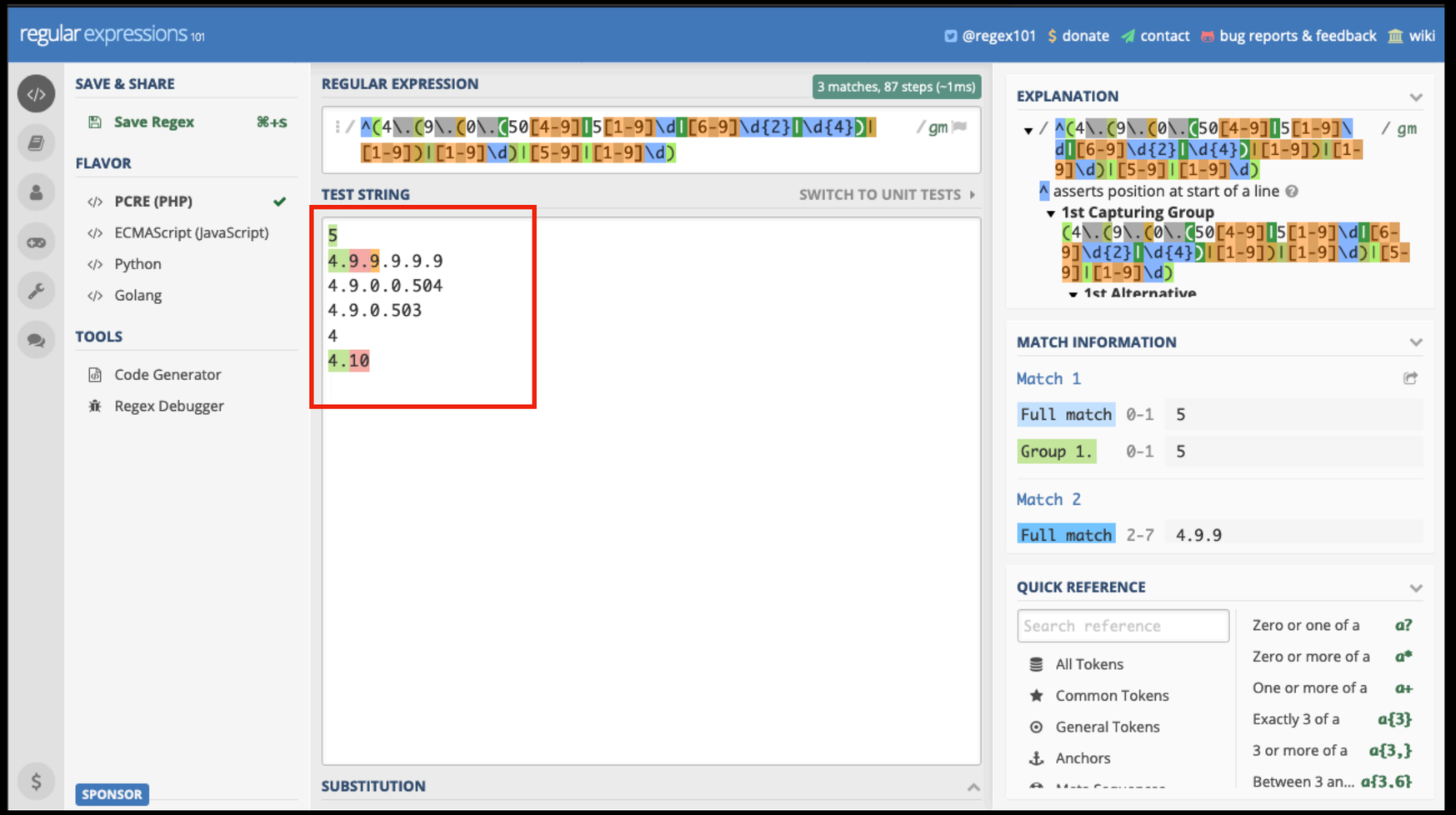
Practice!

https://regex101.com/



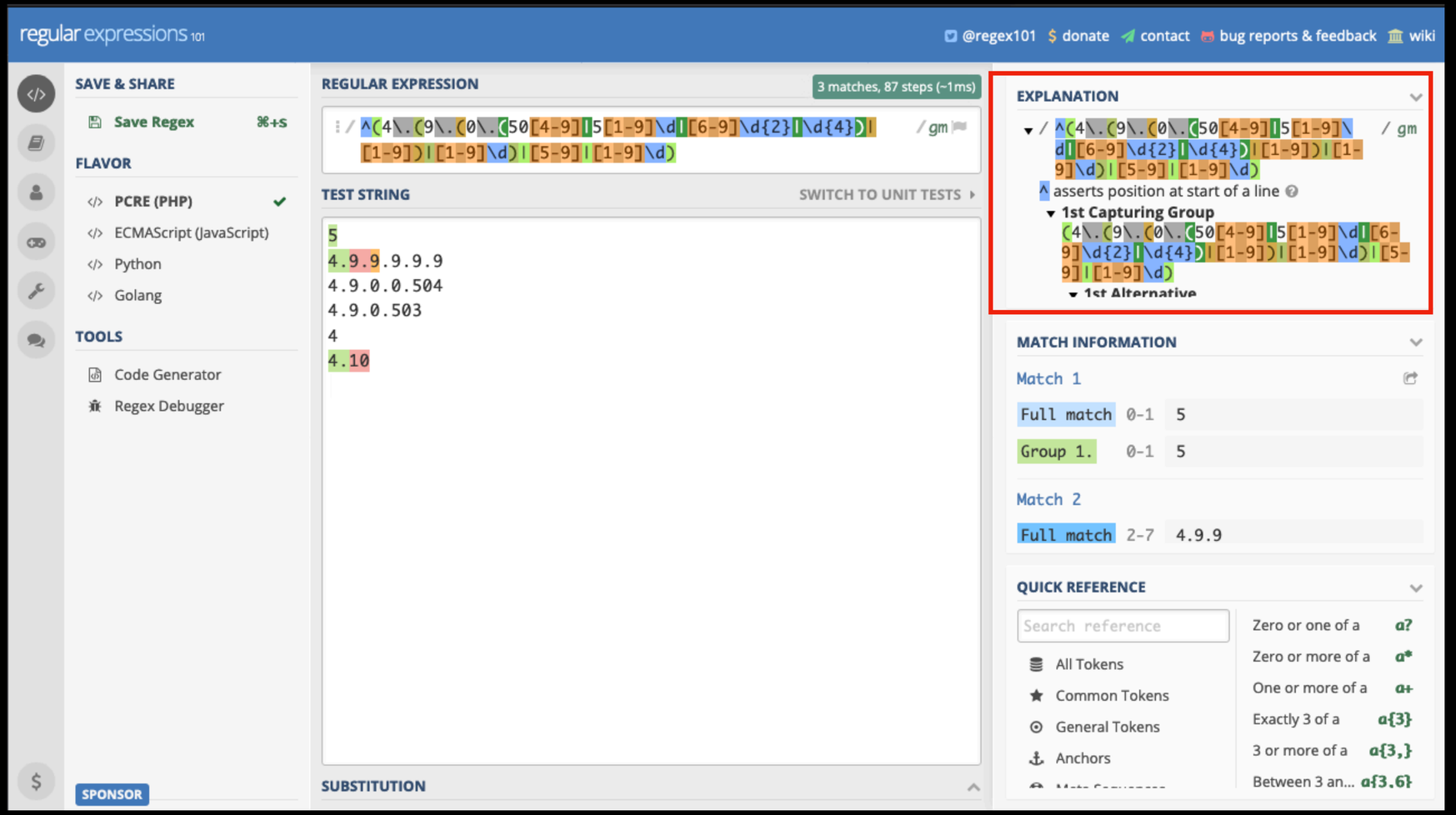
Practice!

https://regex101.com/



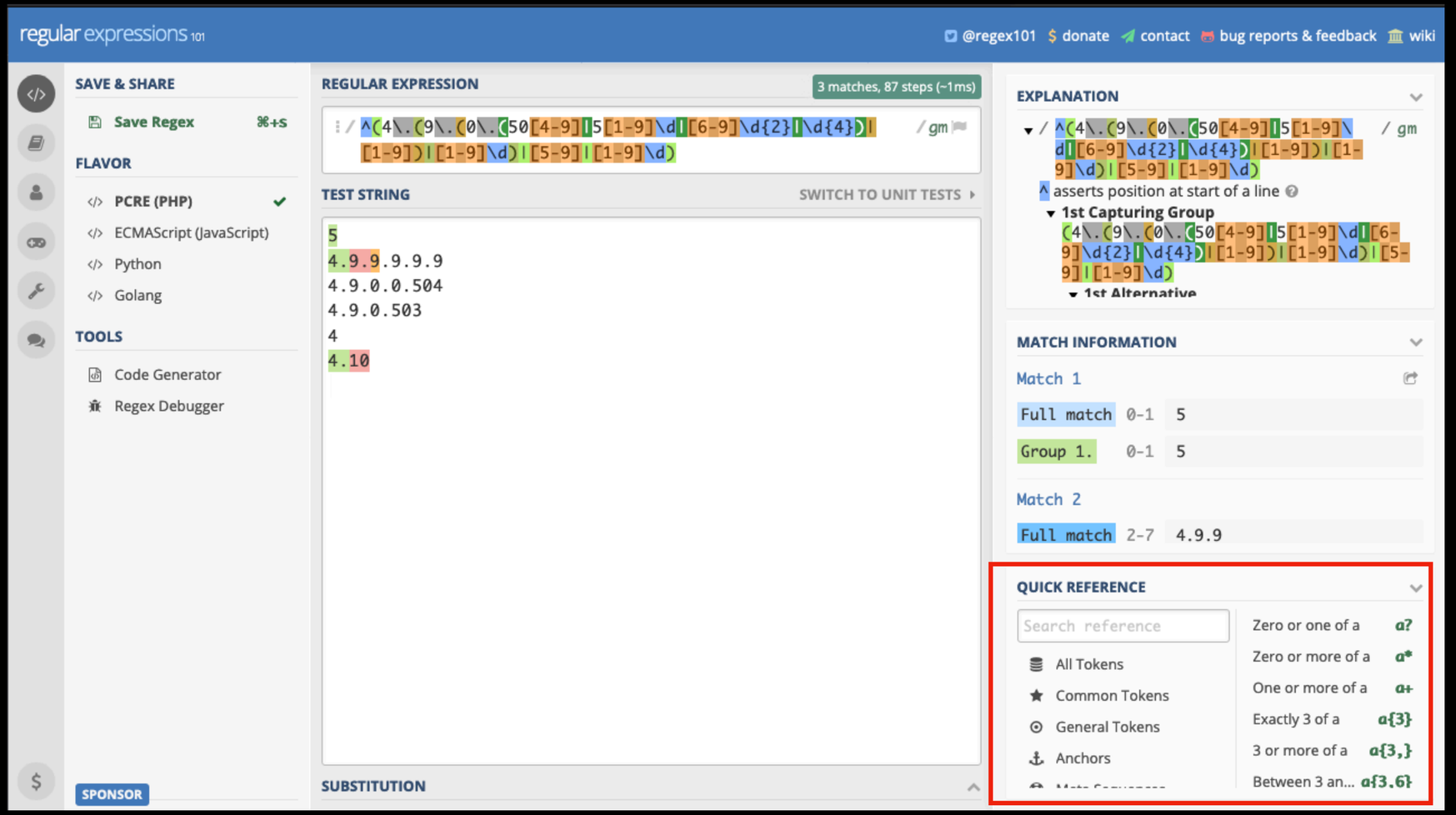
Practice!

https://regex101.com/



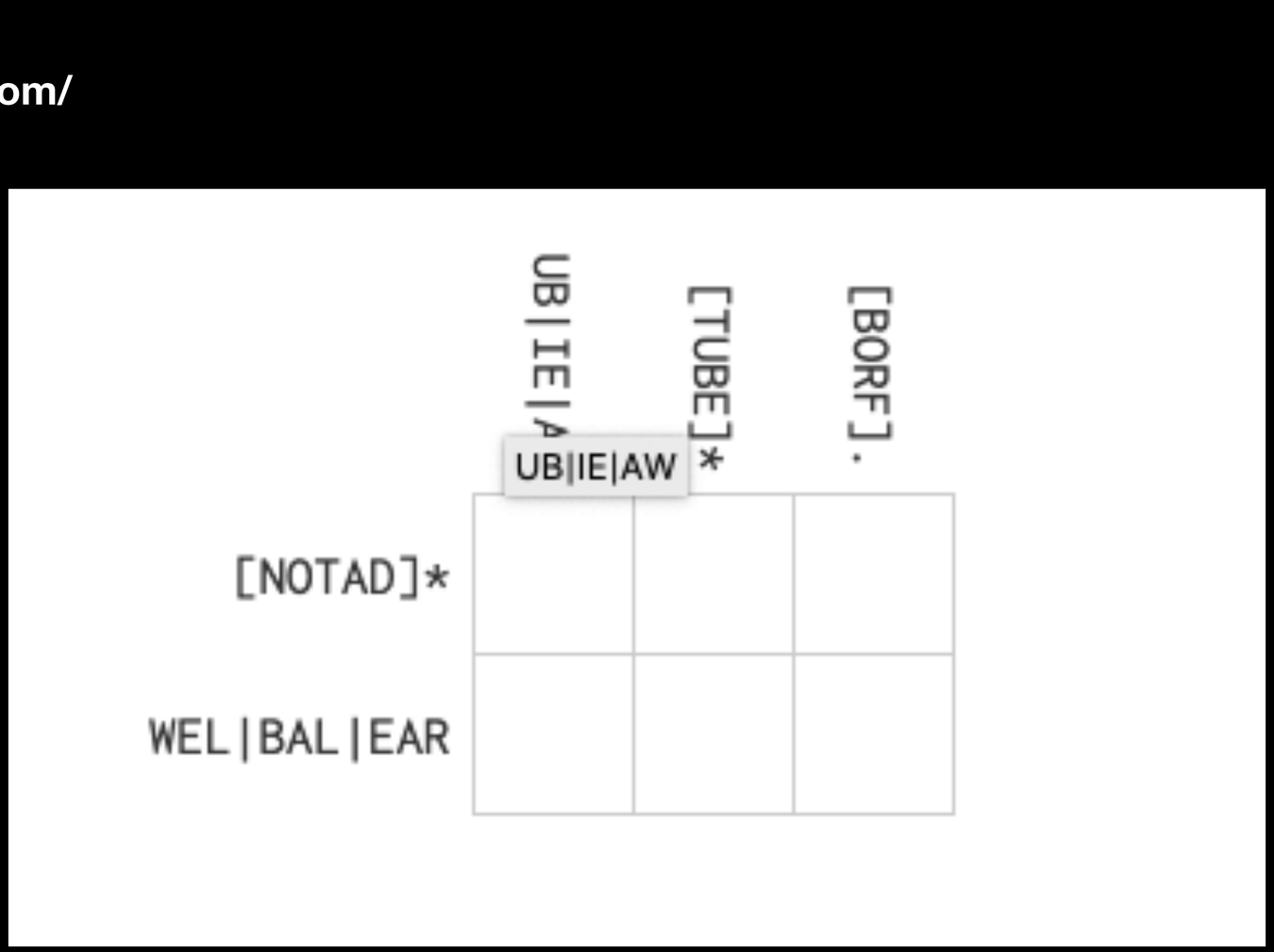
Practice!

https://regex101.com/

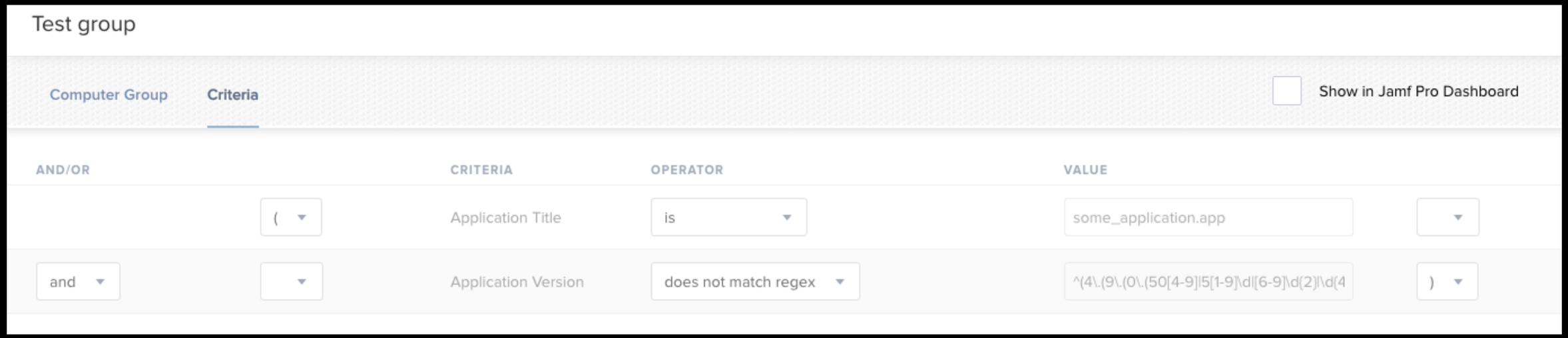


Practice!

https://regexcrossword.com/



Putting regex to work - Jamf



Putting it into practice - python 3

```
>>> import re

>>> result = re.match( r‘^\s*<backupSet id=“(?P<id-
cap>\d+)”>\s*<name>(?P<name-cap>.+)<\/name>’, applog )

>>> result.group(‘id-cap’)
’66’

>>> result.group(‘name-cap’)
‘Mike’s Backup’
```

https://docs.python.org/3/library/re.html

Putting it into practice - zsh

```
% regex="<backupSet id=\"([[:digit:]]+)\">"

% if [[ $applog =~ $regex ]]; then
    echo $MATCH           // <backupSet id="66"
    echo ${match[1]}      // 66
  fi
```

http://zsh.sourceforge.net/Doc/Release/Conditional-Expressions.html

Summary

- Methodology
- Positional anchors - ^ and $
- Alternatives - […]
- Classes - \d, \s, \w
- Quantifiers - {x,y}, ?, +, *
- Groups - (…)

Questions?