Applications

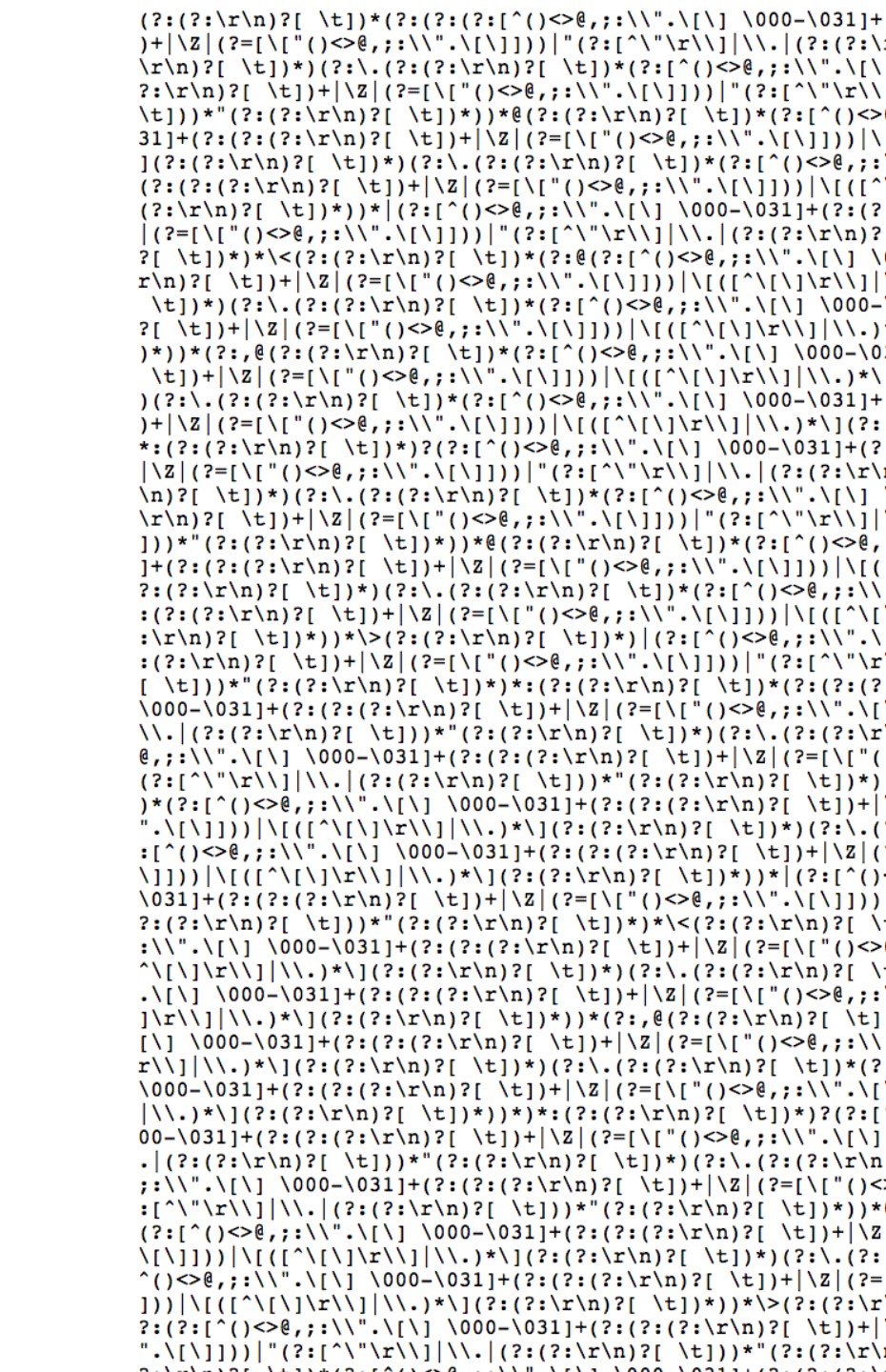
- RE can be used to extract emails, proxies, IPs, phone numbers, addresses, HTML tags, URLs, links, dates, etc.
- AND for even more impressive tasks :
  - Java library for recognizing gestures (ezGestures) !
  - Idea – use regexes to decode gestures and extract them from the stream
  - Transform movements into a series of up/down/left/right actions, represent as a string, e.g. “UDLR”.
  - Use regex to match any combination of actions.

  - SO ! What can we conclude from that?

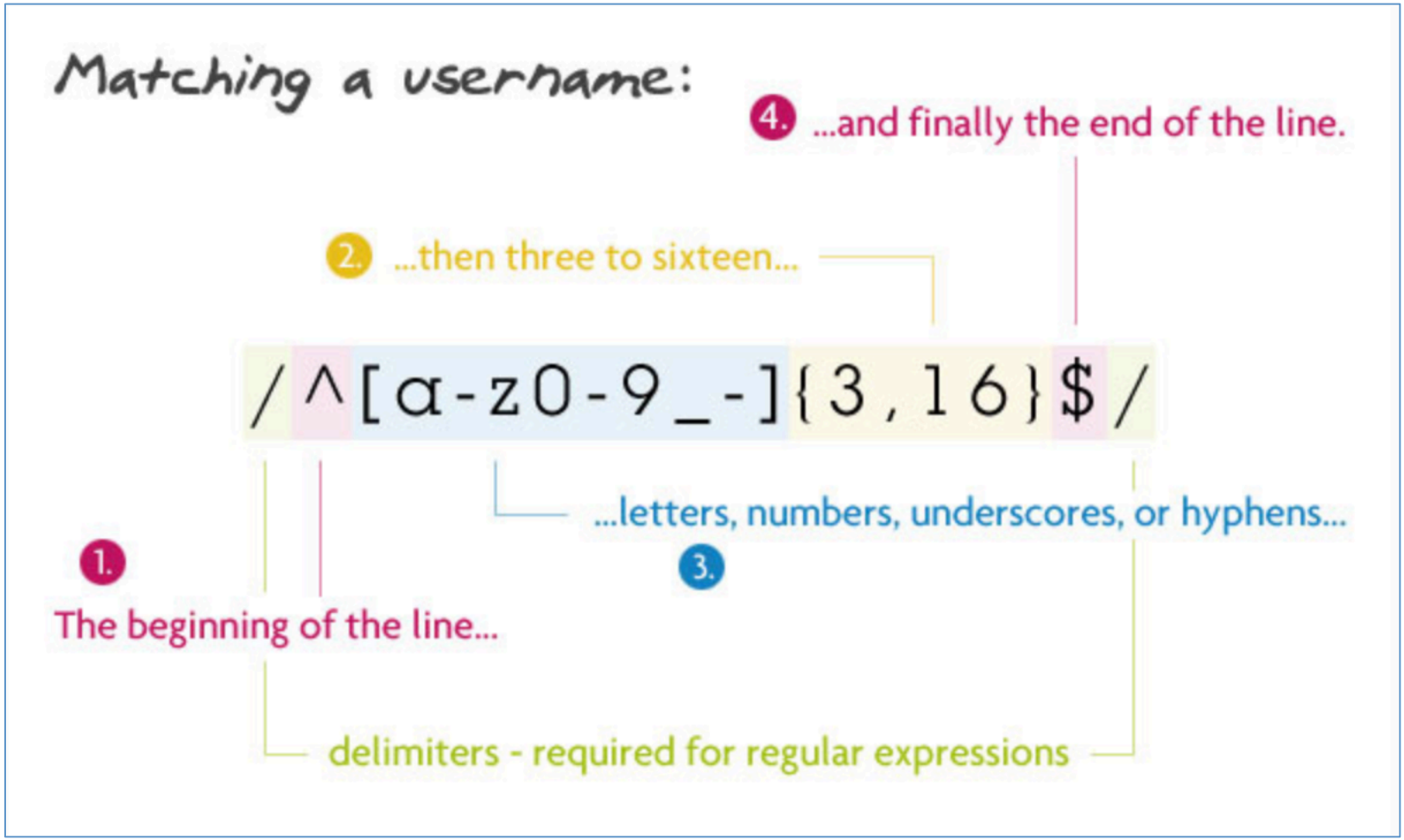
Applications

- If you can translate a sequence of interesting data into a simple alphabetic encoding, you can then use regexes to match and extract patterns from it.

- **So, regex is not just for text !**



Examples

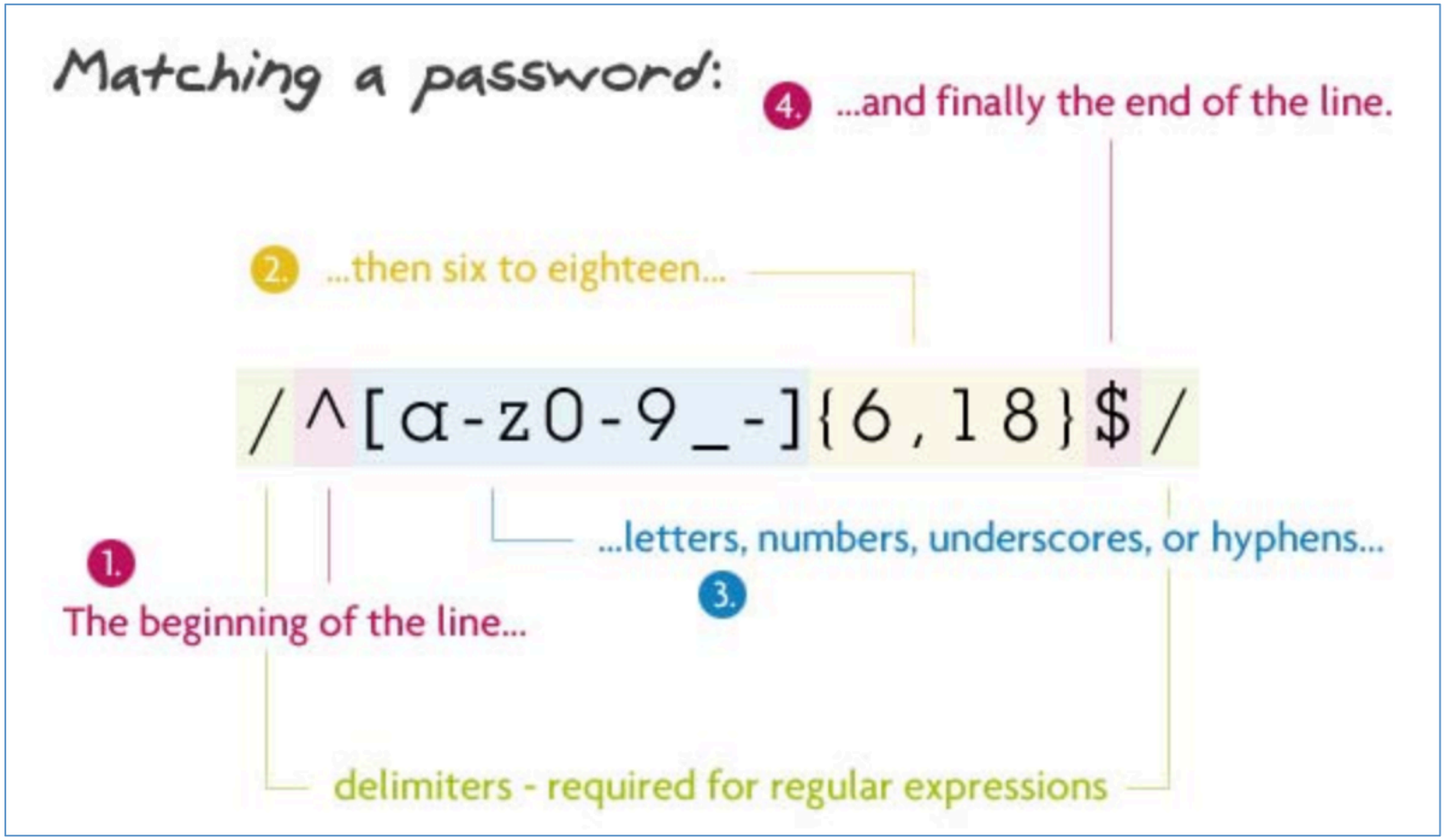


String that matches:

my-us3r_n4m3

String that doesn't match:

th1s1s-wayt00_l0ngt0beausername (too long)


Matching a password:
4. ...and finally the end of the line.
2. ...then six to eighteen...
/^[a-z0-9_-]{6,18}$/
...letters, numbers, underscores, or hyphens...
3.
1.
The beginning of the line...
delimiters - required for regular expressions

Is it good?

What if my requirement is to have a password that contains at least eight characters, including at least one number and includes both lower and uppercase letters and special characters, for example #, ?, !.

?

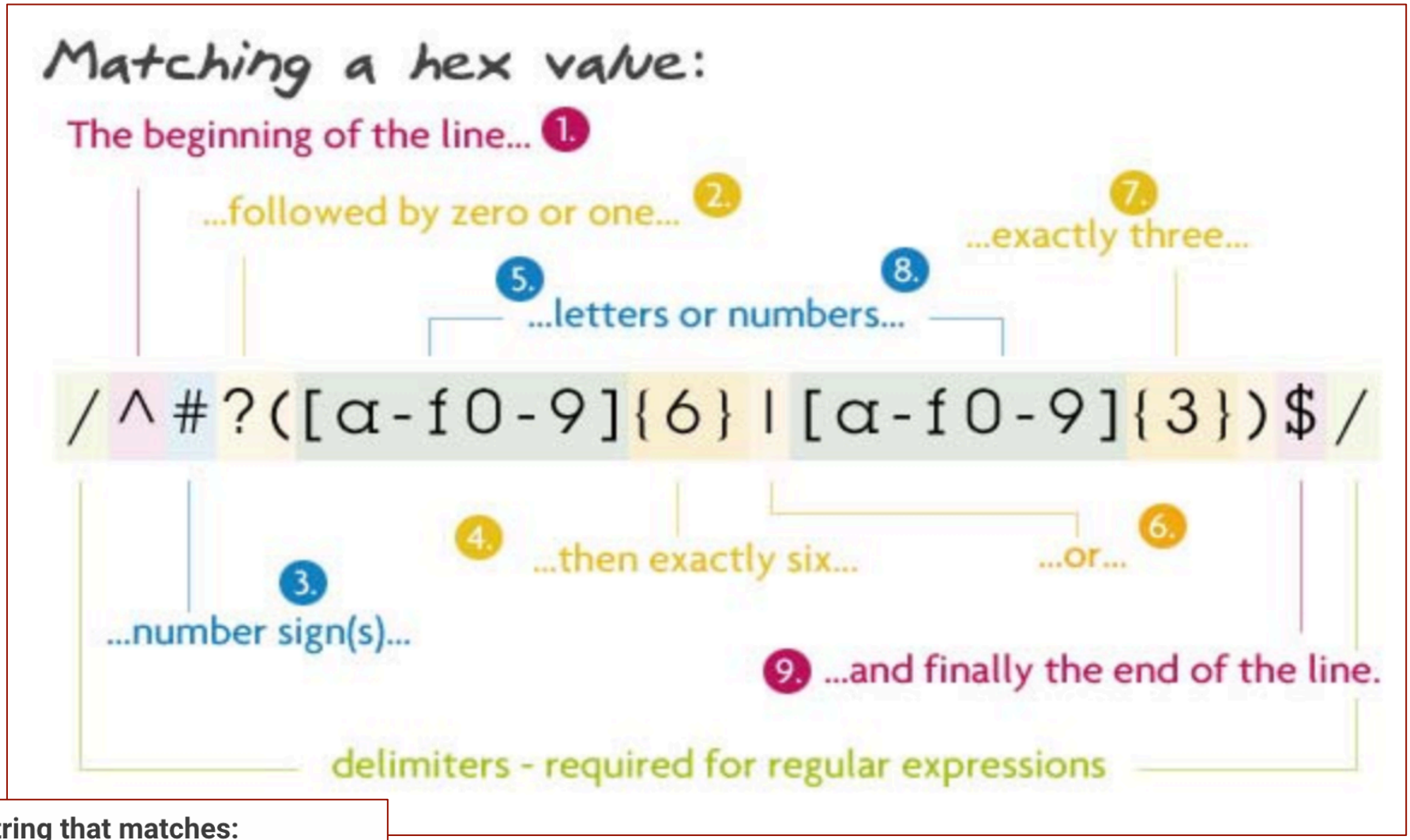
Use positive lookaheads!

e.g. q**(?=**u**)** matches a q that is followed by a u, without making the u part of the match.

```
^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,}$
```

This regex will enforce these rules:

- At least one upper case English letter, `(?=.*?[A-Z])`
- At least one lower case English letter, `(?=.*?[a-z])`
- At least one digit, `(?=.*?[0-9])`
- At least one special character, `(?=.*?[#?!@$%^&*-])`
- Minimum eight in length `.{8,}` (with the anchors)

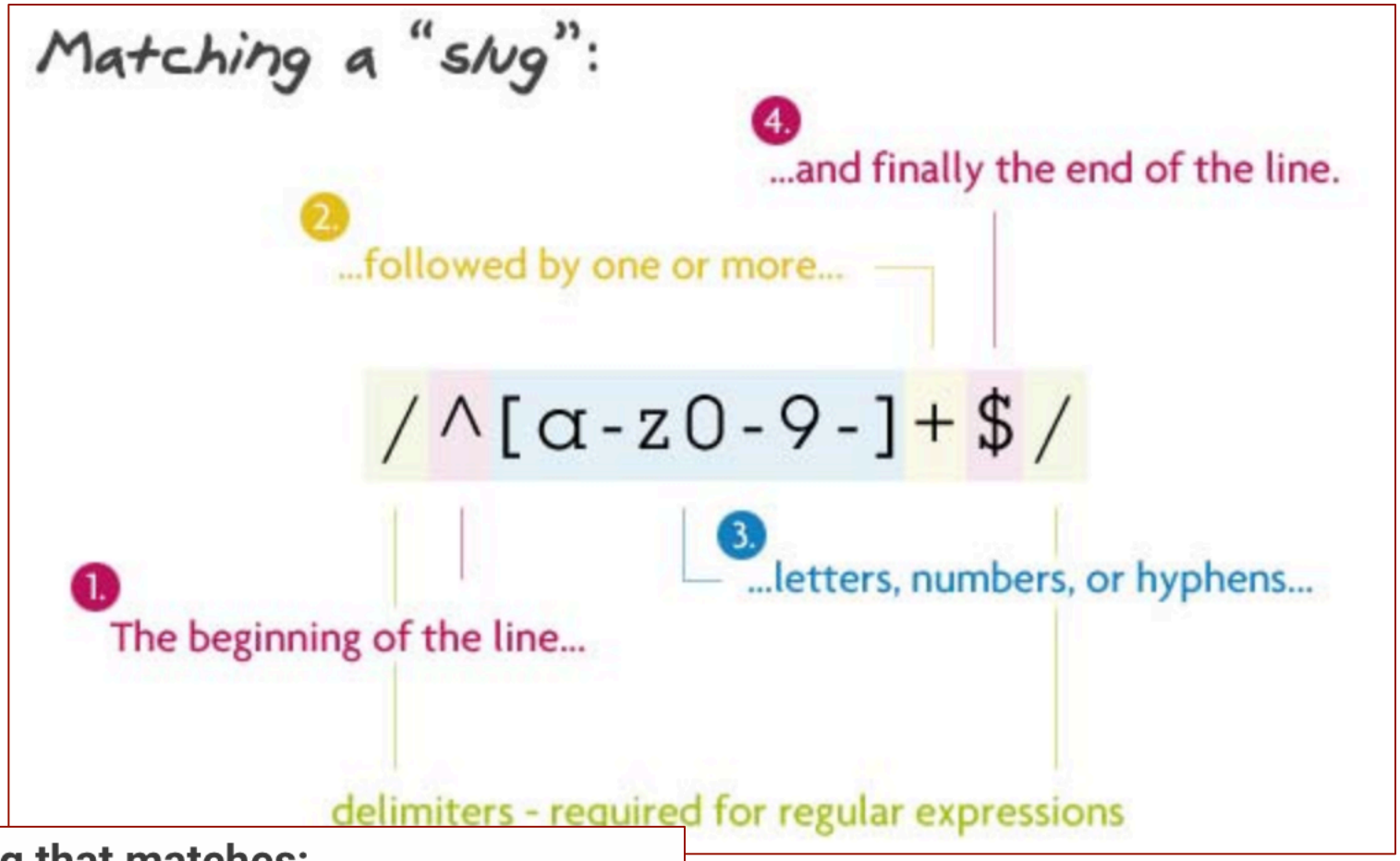


String that matches:

#a3c113

String that doesn't match:

#4d82h4 (contains the letter h)



String that matches:

my-title-here

String that doesn't match:

my_title_here (contains underscores)

You will be using this regex if you ever have to work with mod_rewrite and pretty URL's.

A slug is a part of the URL when you are accessing a resource. Say you have a URL such as the one below, that displays all of the cars in your system:

```
http://localhost/cars
```

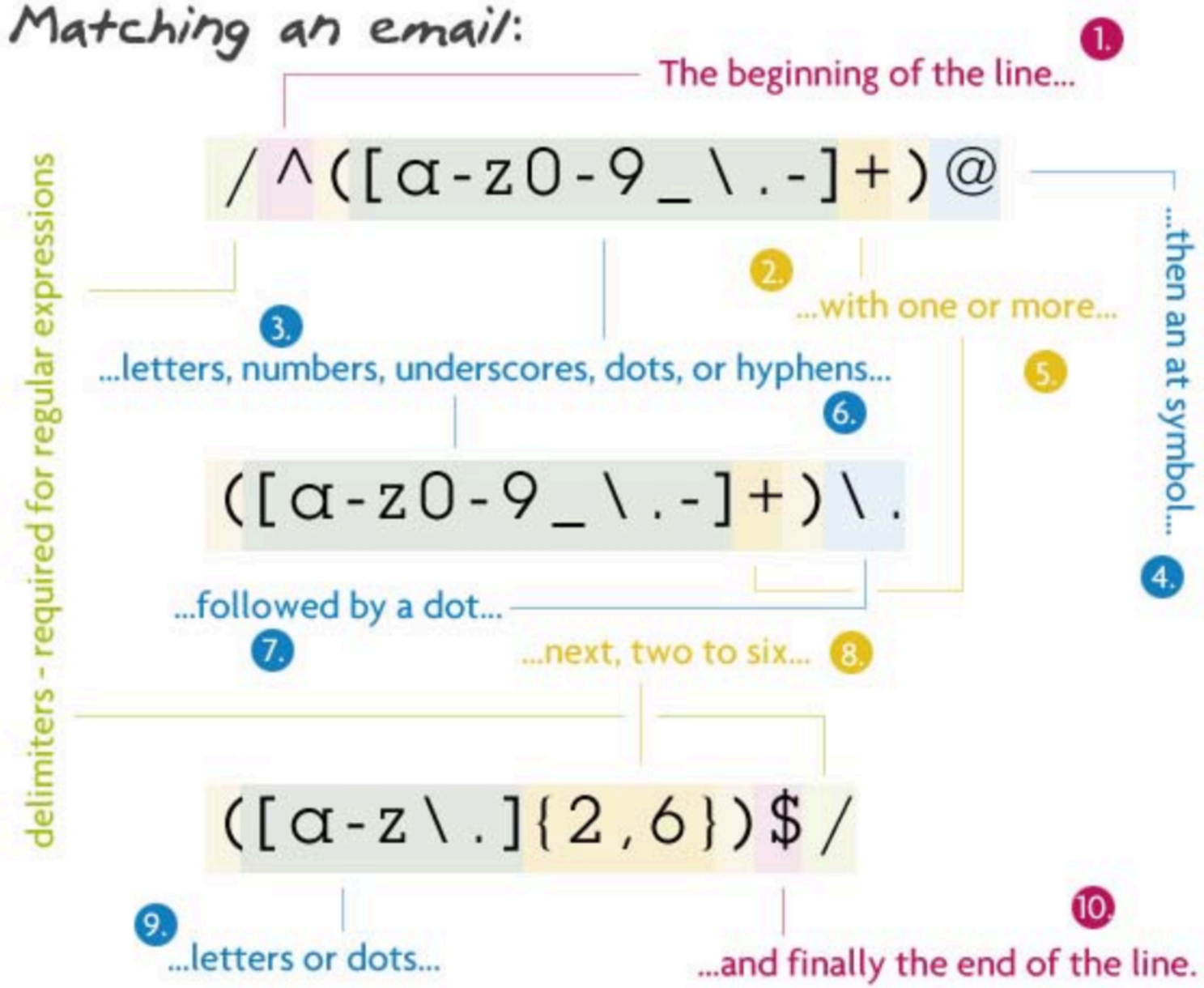
When you would want to reference a particular car in your system, you would provide the following URL:

```
http://localhost/cars/audi-a6/
```

Notice how the URL is still very logical, and very SE friendly. In terms of using the slug, that's at your own discretion. The *audi-a6* string above may be a unique identifier for a car in your system - let's say that you have a relational database with the following fields:

```
id
car_name
car_brand
car_unique_identifier
```

The field car_unique_identifier would then be used in store the values that get displayed in the slug; in the example that I've specified above with an Audi A6 car, this is where your *audi-a6* string would live.


Matching an email:
1.
The beginning of the line...
/^([a-z0-9_\.-]+)@
...then an at symbol...
4.
2.
...with one or more...
5.
3.
...letters, numbers, underscores, dots, or hyphens...
6.
([a-z0-9_\.-]+)\.
...followed by a dot...
7.
...next, two to six...
8.
([a-z\.]{2,6})$/
delimiters - required for regular expressions
9.
...letters or dots...
10.
...and finally the end of the line.

Pattern:

```
/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
```

Description:

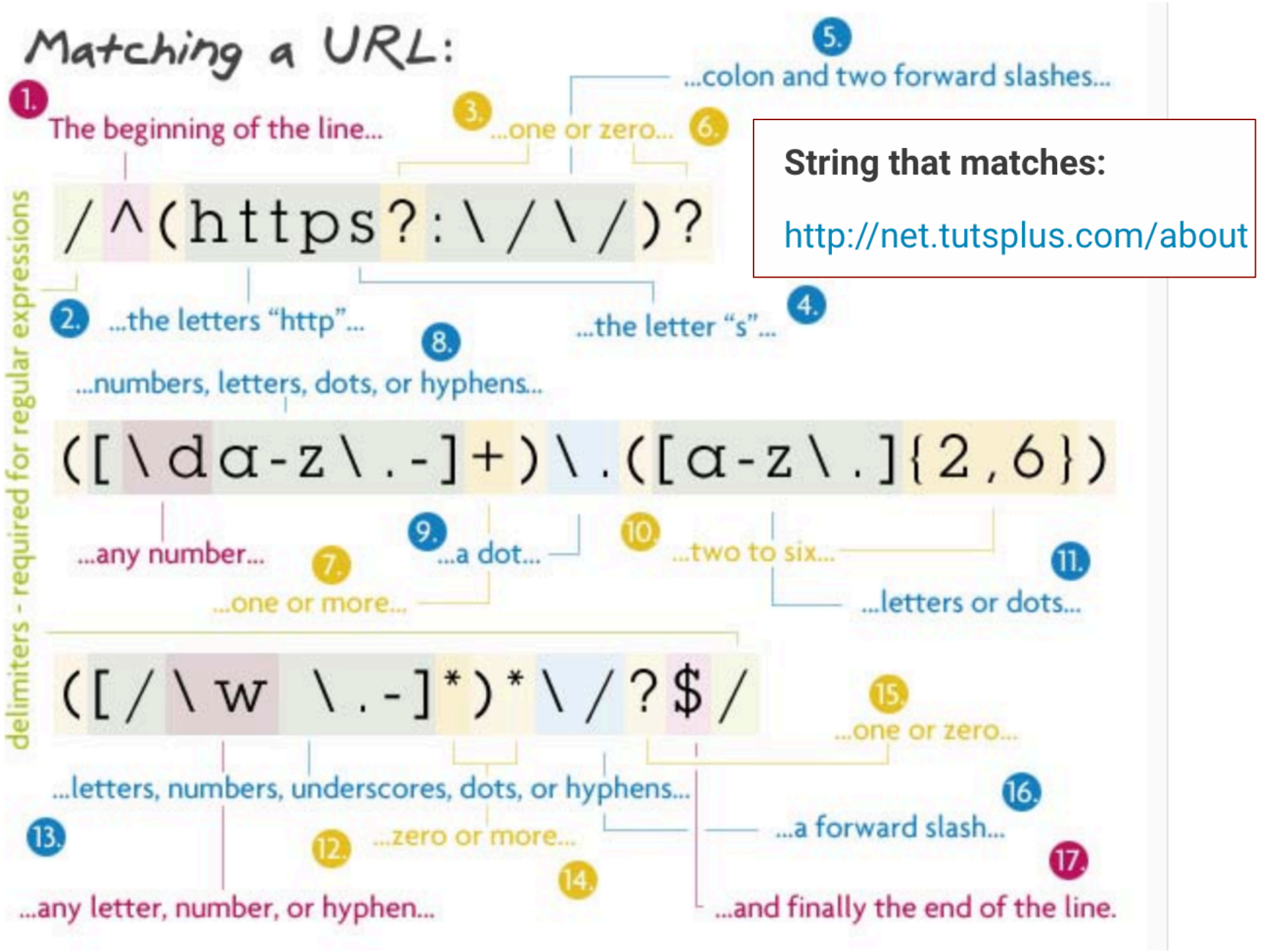
We begin by telling the parser to find the beginning of the string (^). Inside the first group, we match one or more lowercase letters, numbers, underscores, dots, or hyphens. I have escaped the dot because a non-escaped dot means any character. Directly after that, there must be an at sign. Next is the domain name which must be: one or more lowercase letters, numbers, underscores, dots, or hyphens. Then another (escaped) dot, with the extension being two to six letters or dots. I have 2 to 6 because of the country specific TLD's (.ny.us or .co.uk). Finally, we want the end of the string ($).

String that matches:

john@doe.com

String that doesn't match:

john@doe.something (TLD is too long)



String that matches:

http://net.tutsplus.com/about