# REGEX Ellaboration

Regular Expressions (REGEX) are one of the most powerful tools available to web developers for parsing text. They can be used to extract, search and replace data from a large number of sources - including files such as CSV, HTML or JSON. This gist will use one ornate example to show you how they work so, hopefully, become a bit more familiar with REGEX and how they work.

## Summary

As previously mentioned, Regular expressions are a powerful and are used in many different areas of development. This gist will explore how the REGEX pattern used to parse Illinois driver's license numbers is constructed, but will also provide information on the significance of the sequences and characters used to do this through disecting the pattern.

For clarity here is a brief explanation of how an Illion License Number is constructed:

There are 4 parts to IL drivers license numbers. It is of the form SSSS-FFFY-YDDD.

As an example we'll use:

```
John Q Public, a nice man born January 1, 1980

DL number: P142-4758-0001
```

**Part 1** - SSSS : The first part (four characters) is a Soundex Code significant of the License holder's surname. *"P142" in our example.*

**Part 2** - FFF : The second part (four characters) is a numeric value derived from the License holder's first name and middle initial. *"475" in our example.*

**Part 3** - YY : This is the easiest part to pick out. This is a two digit representation of the License holder's birth year. *"8-0" in our example.*

**Part 4** - DDD : This final part is a numeric value derived from the License holder's month and date of birth + gender modifier. *"001" in our example.*

Resulting in **P142-4758-0001** representing John Q Public in the eyes of the IL DMV.[^1] [^2]

---

**So knowing the aforementioned formula, we can create a REGEX pattern that matches a IL Driver's License like this:**

---

```
/^[A-Z]{1}[0-9]{3}-[0-9]{4}-[0-9]{4}$/gm
```

---

## Table of Contents

# Regex Components

## Anchors

What are REGEX Anchors? A regular expression anchor matches at the beginning (^) or end ($) of a line when it appears as one of the first characters on its own line; or at the start and end of each line (either ^$).

In our example:

```
/^[A-Z]{1}[0-9]{3}\-[0-9]{4}\-[0-9]{4}$/gm
```

## Quantifiers

The most basic of all regular expression operators is the quantifier, which typically comes in five flavors: “*”, “+”, “?”, {n}, and {m}.

| Quantifier | Description |
|---|---|
| `x*` | 0 or more repetitions of *x* |
| `x+` | 1 or more repetitions of *x* |
| `x?` | 0 or 1 instances of *x* |
| `x{m}` | **exactly** *m* instances of *x* |
| `x{m+}` | **at least** m instances of *x* |
| `x{m,n}` | between *m* and *n* (inclusive) instances of *x* |
| Citation | MIT.edu PDF |

In our example: `/^[A-Z]{1}[0-9]{3}\-[0-9]{4}\-[0-9]{4}$/gm`

So in the above example, we are saying that we want the first character to be a capital letter, and subsequent 3 characters to be numbers, followed by a "-"

## Grouping Constructs

As regular expressions grow more complex, you may check multiple parts of a string to determine that different sections fullfil different requirements. To break these sections up, use grouping constructs like parentheses (). Each section within parentheses are known as 'subexpression'. By placing a part of the regular expression inside round brackets or parentheses, you can group that part and apply quantifiers to

it. This includes limits on alternation so only specific parts are affected by whatever change is being made in your regex.

A very good example for articulating this point would be finding `anchor` tags in an HTML document. To do this you would define the regular expression pattern as follows:

/(<a href=)(["'])(.?)(["'])> (.?)(</a>)/gmi

Would find both:

<a href='trashparty.xyz'>TrashParty</a>

and

<a href="trashparty.xyz">Trash Party</a>

Side Note: *TrashParty may be the single greatest web application ever created. You should check it out.*

## Bracket Expressions

Bracket expressions are powerful tools for searching and selecting data. They don't require a string to meet every requirement in the pattern, they just need one of them! This means that [a-z0-9_-] searches for alphanumeric characters or two special symbols included within brackets (hyphens (-) and underscores (_)).

- [ ] we're able to define what's called a character class - this defines anything with those letters- which can make finding things much easier!

Our example is riddled with grouping, but here's a simple example: The regular expression `/ch[eiou]ck/igm` would find the following:
`The chick is rich, I mean chock full of checks.Sometimes she even chucks them at cashiers to show them she is rich.`

Now, that sentence was garbage, but it made for a good example.

## Character Classes

A character class in a regex defines a set of characters, any one of which can occur in an input string to fulfill the match. The bracket expressions outlined previously--including positive and negative groups--are considered character classes.

Some other common ones are:

| Syntax | Description |
|---|---|
| . | Wildcard (matches any character) |
| \d | Matches any digit 0123456789 |
| \w | Matches "word" (letters, digits, and _) |
| \W | Matches non-word characters |
| \t | Matches tab characters |

| Syntax | Description |
| --- | --- |
| \r | Matches return |
| \S | Matches non-whitespace |
| Citation | MIT.edu PDF |

So in our example:

This ➡️ `/^[A-Z]{1}[\d]{3}\-[\d]{4}\-[\d]{4}$/gm`

is the same as

This ➡️ `/^[A-Z]{1}[0-9]{3}\-[0-9]{4}\-[0-9]{4}$/gm`

## The OR Operator

Using the OR operator "|" in a regex pattern, tells the engine to look for multiple sets of criteria, but not mix. *i.e The pattern `/^cat|dog/gmi` would find `cats play with yarn, but dogs don't.` as well as

`CATS are terrible, DOGS RULE!`

*. . . but I thought REGEX was case-sensitive. . . . .*

Which brings us to Flags

## Flags

To make regular expressions more versatile, there are six[^3] optional flags that can be used. These include:

- /i for case-insensitive matching
- /m Multi-line Search
- /g which tells the engine to find all occurrences of pattern in one or many strings instead of stopping at first match only (/g stands for global)
- /s allows . to match newline characters
- /u tells the engine to treat a pattern sequence as a string of unicode characters
- /d generates indices for substring matches

In our example, we are assuming you're looking for all occurrences and on multiple lines.

`/^[A-Z]{1}[\d]{3}\-[\d]{4}\-[\d]{4}$/gm`

## Character Escapes

A backslash in a regex can be used to escape characters that would otherwise have special meaning. For example, the open curly brace is usually used as an indication of quantifiers but adding a backslash before it ({) means that you are looking for the character and not beginning another type of identifier such as "{"

A slash in your regular expression signifies ending one set or pattern and starting another. There's no need to use this when searching strings with certain symbols within them though, because there will always only

ever be one instance per string which needs escaping - so using { instead leaves more room behind for other identifiers like "+" if needed: {+

In our example:

```
/^[A-Z]{1}[0-9]{3}\-[0-9]{4}\-[0-9]{4}$/gm
```

## Author

A web development student. Peter loves learning new things and developing them into useful other things for people. His strengths lie in JS & CSS and seeing the "big picture" of almost everything in life. As a student of the world, he is committed to **always** learning.

Please, feel free to contact ✉️ Peter via email or visit his :octocat Github profile.

---

### Footnotes

1. *For more in depth analysis of how driver's license numbers work* click here

2. *For a fun generator and analyzer* click here

3. According to MDN there is an additional "y" flag that indicates a "sticky" search. There site says that there are 6 REGEX flags, but then proceed to show you a chart with 7 rows