Regex character classes

A character class is something in between those two extremes. A character class is a list of characters that can be matched.

The list is placed in square brackets [].

For example [abc] will match either 'a' or 'b' or 'c'.

Just as a regular character or a . can match exactly one character so does a character class. Later we are going to learn about quantifiers that will allow us to say how many of something we would like to match, but for now remember that a character class always matches exactly one character. If it cannot fulfill the match then the whole regex matching fails.

So what if we have a bunch of strings and we would like to make sure only strings containing any of the following will match? #a#, #b#, #c#, #d#, #e#, #f#, #@# or #.# That is, we would like the string to have a # character, followed by 'a', 'b', 'c', 'd', 'e', 'f', '@', or '.', followed by another # character. (We are using # in this example in order to get you used to seeing 'strange' characters that have no special meaning.)

The regex that will match those looks like this: /#[abcdef@.]#/.

It says: match a #, then match any one(!) of the characters in the square bracket, then match another #.

this will match

```
"#a#"

"ab #z#a# "

"ab #.# "
```

but will not match any of the following:

```
"ab #q# "

"ab ## "

"##"

"#ab#"

"#aa#"

"# #"

"###"

"#-#"
```

Two notes:

- The regex won't match "##" or "#ab#" because the character class must match exactly one character between the two '#' characters.
- The '.' inside the character class lost its special meaning of "everything except newline" and can match a single '.' only.

In general, most special characters lose their special meaning inside a character class, but there are of course exceptions. There are even character that gain special meaning inside a character class.

Range in a character class

Programmers are lazy typing in all the characters between 'a' and 'f' in the regex /#[abcdef@.]#/ was really tiring. If we had to type in all the characters between 'a' and 'z' that would be even worse and it would be very error-prone. What if I miss one of the characters? Instead of that regexes allow us to define a range of characters from the ASCII table using a dash (-). The previous regex could be written as /#[a-f@.]#/

So as you can see a dash -, that did not have any special meaning outside of a character class, inside has the special "range-making" meaning.

Of course you will then want to know how can you express that one of the characters you'd like to match in the character class is a dash, and the answer is that if you place the dash as the first or the last character in the character class, then it will be just a plain dash. So /#[a-f@.-]#/ will match all the above and also "#-#".

Another frequently asked question at this point is how to include a closing square bracket] in a character class. That's simple too. You just need to "escape" it be preceding with a back-slash: \].

Negated character class

What if we would like to allow the matching of any character between two '#' characters **except** 'a', 'b', or 'c'? We would need to construct a character class with all the characters in the world and Unicode has more that 110,000 characters. That would be a lot of work to type in. Instead of that, Perl allows us to negate a character class. If we put a Caret (^) as the first character in the character class it will mean the character class can match any one character except those mentioned in the character class. So [^abc] would match exactly one character that is not 'a', nor 'b', nor 'c'. Our full regex then would look like /#[^abc]#/.

This regex will match these strings:

```
"abc #z# z"

"#z#"
```

but will **not** match any of these strings:

```
"abc #a# z"

"#xyz#"

"##"
```

Note, it won't match the string '##' or the string "#xyz#", because the negated character class still has to match exactly one character.

Summary

```
/a[bc]a/      # aba, aca

/a[2#=x?.]a/  # a2a, a#a, a=a, axa, a?a, a.a

              # inside the character class most of the spec characters
lose their

              # special meaning  BUT there are some new special
characters

/a[2-8]a/     # is the same as /a[2345678]a/

/a[2-]a/      # a2a, a-a        - has no special meaning at the ends

/a[-8]a/      # a8a, a-a

/a[6-C]a/     # a6a, a7a ... aCa

              #     characters from the ASCII table: 6789:;<=>?@ABC
but this is not recommended, don't use it!

/a[C-6]a/     # syntax error


/a[^xy]a/     # "aba", "aca"  but not "aya", "axa" and remember, not
"aa"

              # ^ as the first character in a character class means

              # a character that is not in the list

/a[b^x]a/     # aba, a^a, axa,  but not aza
```