LESSON 3

Matching Sets of Characters

In this lesson you'll learn how to work with sets of characters. Unlike the `.`, which matches any single character (as you learned in the previous lesson), sets enable you to match specific characters and character ranges.

Matching One of Several Characters

As you learned in the previous lesson, `.` matches any one character (as does any literal character). In the final example in that lesson, `.a` was used to match both `na` and `sa`, `.` matched both the `n` and `s`. But what if there was a file (containing Canadian sales data) named `ca1.xls` as well, and you still wanted to match only `na` and `sa`? `.` would also match `c`, and so that filename would also be matched.

To find `n` or `s` you would not want to match *any* character, you would want to match just those two characters. In regular expressions a set of characters is defined using the metacharacters `[` and `]`. `[` and `]` define a character set, everything between them is part of the set, and any one of the set members must match (but not all).

Here is a revised version of that example from the previous lesson:

TEXT

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

REGEX

```
[ns]a.\.xls
```

RESULT

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

ANALYSIS

The regular expression used here starts with `[ns]`; this matches either `n` or `s` (but not `c` or any other character). `[` and `]` do not match any characters—they define the set. The literal `a` matches `a`, `.` matches any character, `\.` matches the `.`, and the literal `xls` matches `xls`. When you use this pattern, only the three desired filenames are matched.

Note Actually, `[ns]a.\.xls` is not quite right either. If a file named `usa1.xls` existed, it would match, too. The solution to this problem involves position matching, which will be covered in Lesson 6, "Position Matching."

Tip As you can see, testing regular expressions can be tricky. Verifying that a pattern matches what you want is pretty easy. The real challenge is in verifying that you are not also getting matches that you don't want.

Character sets are frequently used to make searches (or specific parts thereof) not case sensitive. For example:

TEXT

```
The phrase "regular expression" is often
abbreviated as RegEx or regex.
```

REGEX

```
[Rr]eg[Ee]x
```

RESULT

```
The phrase "regular expression" is often
abbreviated as RegEx or regex.
```

ANALYSIS

The pattern used here contains two character sets: `[Rr]` matches `R` and `r`, and `[Ee]` matches `E` and `e`. This way, `RegEx` and `regex` are both matched. `REGEX`, however, would not match.

Tip If you are using matching that is not case sensitive, this technique would be unnecessary. This type of matching is used only when performing case-sensitive searches that are partially not case sensitive.

Using Character Set Ranges

Let's take a look at the file list example again. The last used pattern, `[ns]a.\.xls`, has another problem. What if a file was named `sam.xls`? It, too, would be matched because the `.` matches all characters, not just digits.

Character sets can solve this problem as follows:

TEXT

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
```

```
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

REGEX

```
[ns]a[0123456789]\.xls
```

RESULT

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

ANALYSIS

In this example, the pattern has been modified so that the first character would have to be either `n` or `s`, the second character would have to be `a`, and the third could be any digit (specified as `[0123456789]`). Notice that file `sam.xls` was not matched, because `m` did not match the list of allowed characters (the 10 digits).

When working with regular expressions, you will find that you frequently specify ranges of characters (`0` through `9`, `A` through `Z`, and so on). To simplify working with character ranges, regex provides a special metacharacter: `-` (hyphen) is used to specify a range.

Following is the same example, this time using a range:

TEXT

```
sales1.xls
orders3.xls
sales2.xls
```

```
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

REGEX

```
[ns]a[0-9]\.xls
```

RESULT

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

ANALYSIS

Pattern `[0-9]` is functionally equivalent to `[0123456789]`, and so the results are identical to those in the previous example.

Ranges are not limited to digits. The following are all valid ranges:

- `A-Z` matches all uppercase characters from `A` to `Z`.
- `a-z` matches all lowercase characters from `a` to `z`.
- `A-F` matches only uppercase characters `A` to `F`.
- `A-z` matches all characters between ASCII `A` to ASCII `z` (you should probably never use this pattern, because it also includes characters such as `[` and `^`, which fall between `Z` and `a` in the ASCII table).

Any two ASCII characters may be specified as the range start and end. In practice, however, ranges are usually made up of some or all digits and some or all alphabetic characters.

Tip - When you use ranges, be careful not to provide an end range that is less than the start range (such as `[3-1]`). This will not work, and it will often prevent the entire pattern from working.

Note - (hyphen) is a special metacharacter because it is only a metacharacter when used between `[` and `]`. Outside of a set, - is a literal and will match only `-`. As such, `-` does not need to be escaped.

Multiple ranges may be combined in a single set. For example, the following pattern matches any alphanumeric character in uppercase or lowercase, but not anything that is neither a digit nor an alphabetic character:

```
[A-Za-z0-9]
```

This pattern is shorthand for

```
 [ABCDEFGHIJKLMNOPQRSTUVWXYZabcde
➥fghijklmnopqrstuvwxyz01234567890]
```

As you can see, ranges makes regex syntax much cleaner.

Following is one more example, this time finding RGB values (colors specified in a hexadecimal notation representing the amount of red, green, and blue used to create the color). In Web pages, RGB values are specified as #000000 (black), #FFFFFF (white), #FF0000 (red), and so on. RGB values may be specified in uppercase or lowercase, and so #FF00ff (magenta) is legal, too. Here's the example:

TEXT

```
<BODY BGCOLOR="#336633" TEXT="#FFFFFF"
      MARGINWIDTH="0" MARGINHEIGHT="0"
      TOPMARGIN="0" LEFTMARGIN="0">
```

REGEX

```
#[0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f]
➥[0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f]
```

RESULT

```
<BODY BGCOLOR="#336633" TEXT="#FFFFFF"
      MARGINWIDTH="0" MARGINHEIGHT="0"
      TOPMARGIN="0" LEFTMARGIN="0">
```

ANALYSIS

The pattern used here contains `#` as literal text and then the character set `[0-9A-Fa-f]` repeated six times. This matches `#` followed by six characters, each of which must be a digit or `A` through `F` (in either uppercase or lowercase).

"Anything But" Matching

Character sets are usually used to specify a list of characters of which any must match. But occasionally, you'll want the reverse—a list of characters that you don't want to match. In other words, *anything but the list specified here*.

Rather than having to enumerate every character you want (which could get rather lengthy if you want all but a few), character sets can be negated using the `^` metacharacter. Here's an example:

TEXT

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
```

```
na1.xls
na2.xls
sa1.xls
ca1.xls
```

REGEX

```
[ns]a[^0-9]\.xls
```

RESULT

```
sales1.xls
orders3.xls
sales2.xls
sales3.xls
apac1.xls
europe2.xls
sam.xls
na1.xls
na2.xls
sa1.xls
ca1.xls
```

ANALYSIS

The pattern used in this example is the exact opposite of the one used previously. `[0-9]` matches all digits (and only digits). `[^0-9]` matches anything by the specified range of digits. As such, `[ns]a[^0-9]\.xls` matches `sam.xls` but not `na1.xls`, `na2.xls`, or `sa1.xls`.

Note ^ negates all characters or ranges in a set, not just the character or range that it precedes.

Summary

Metacharacters [and] are used to define sets of characters, any one of which must match (OR in contrast to AND). Character sets may be enumerated explicitly or specified as ranges using the - metacharacter. Character sets may be negated using ^; this forces a match of anything but the specified characters.