Patterns Tutorial

If you are new to strings and patterns, reading the StringsTutorial and StringLibraryTutorial may be useful. Please note this tutorial is an introduction to the concepts involved in patterns and some examples of the their use. It is not an advanced treatment of pattern matching. You would be very wise to read section 5.4 of the Reference Manual [1] for more information. For practical examples of string and pattern usage, have a look at StringRecipes.

See also: http://www.wowwiki.com/HOWTO:_Use_Pattern_Matching

Introduction to patterns

The Lua string library has some very useful and flexible functionality for string manipulation. The flexibility of these functions lies in part to the use of *patterns*, sometimes (erroneously) called *regular expressions*. This is a way of specifying a character pattern to look for in a string. In the following example we just look for a (character) pattern `'an'` in the string `'banana'`. The `string.find()` function returns the position of the first occurrence of `'an'` in `'banana'`.

```
> = string.find('banana', 'an')   -- find 1st occurance of 'an'
2       3
> = string.find('banana', 'lua')  -- 'lua' will not be found
nil
```

This is pretty useful but a little rigid, e.g. what if we wanted to look for some four letter words that begin with "w":

```
> text = 'a word to the wise'
> = string.sub(text, string.find(text, 'w...'))
word
> = string.sub(text, string.find(text, 'w...', 5)) -- bit further on...
wise
```

In the above example we use `string.sub()` to display the substring which `string.find()` found us. The search pattern we used both times was `'w...'`. The '.' represents a wildcard character, which can represent any character. Patterns contain sequences of normal characters and pattern formatters which have special meanings.

findpattern()

We'll define a useful function for this tutorial. This function just takes a string and a pattern and returns the substring which matches the pattern in that string:

```
> function findpattern(text, pattern, start)
>>  return string.sub(text, string.find(text, pattern, start))
>> end
>
> = findpattern('a word to the wise', 'w...')
word
> = findpattern('a word to the wise', 'w...', 5)
wise
```

Character classes

A *character class* represents a set of characters. The wildcard '.' character can represent any character. There are several other classes representing subsets of characters, e.g. numbers, letters, uppercase characters, lowercase characters etc. These sets have the format *%X* where *X* is a letter representing the class. All the character classes available are fully documented in the Reference Manual.[2] Let's see a few examples:

```
> = findpattern('The quick brown fox', '%a') -- %a is all letters
T
> = findpattern('it is 2003', '%d')          -- %d finds digits
2
> = findpattern('UPPER lower', '%l')         -- %l finds lowercase characters
l
> = findpattern('UPPER lower', '%u')         -- %u finds uppercase characters
U
```

Just as we can look for strings in strings, we can look for sequences of characters from the character classes, and we can mix these with regular strings, e.g.,

```
> = findpattern('UPPERlower', '%u%l')   -- upper followed by lower
Rl
> = findpattern('123 234 345', '%d3%d') -- digit 3 digit
234
```

Sets

As well as the predefined character classes, of the form *%X*, we can also explicitly define our own *sets*. These are represented as *[set]* where "set" is a list of characters to look for.

```
> = findpattern('banana', '[xyjkn]') -- look for one of "xyjkn"
n
```

Use the hyphen '-' to denote a range of characters.

```
> = findpattern('banana', '[j-q]') -- equivalent to "jklmnopq"
n
```

We can use character classes in sets:

```
> = findpattern('it is 2003', '[%dabc]')
2
```

Sets can also be used in the form *[^set]* which means find any of the characters *not* in the set listed, e.g.,

```
> = findpattern('banana', '[^ba]')       -- we don't want a or b
n
> = findpattern('bananas', '[^a-n]')     -- forget a to n
s
> = findpattern('it is 2003', '[^%l%s]') -- not a lowercase or space
2
```

Repetition

Sometimes we want to look for patterns and we don't know the number of characters in them. Support for searching for variable length patterns is supplied using the characters: `*`, `+`, `-` and `?`. These are described well in the Reference Manual so we'll give a quick summary and some examples here:

- `*` looks for 0 or more repetitions of the previous pattern element, e.g.,

```
> = findpattern('bananas', 'az*')
a
> = findpattern('bananas', 'an*')
```

```
an
> = findpattern('bannnnnanas', 'an*')
annnnn
```

- `+` looks for 1 or more repetitions of the previous pattern element, e.g.,

```
> = string.find('banana', 'az+')   -- won't be found
nil
> = findpattern('bananas', 'an+')
an
> = findpattern('bannnnnanas', 'an+')
annnnn
```

- `-` looks for 0 or more repetitions of the previous pattern element. But, where this differs from `*` is that it looks for the *shortest* sequence of elements, whereas `*` looks for the longest sequence, e.g.,

```
> = findpattern('bananas', 'az-')
a
> = findpattern('bananas', 'an-')
a
> = findpattern('bannnnanas', 'an-')
a
```

- `?` looks for 0 or 1 occurrences of the previous pattern element.

```
> = findpattern('bananas', 'az?')
a
> = findpattern('bananas', 'an?')
an
> = findpattern('bannnnanas', 'an?')
an
```

Captures

Captures can be marked in a pattern using brackets, e.g., `"w(...)"` would capture the last three letters of the four letter word we used in a previous example:

```
> = string.find('a word to the wise', 'w(...)')    -- last three letters
3       6       ord
> = string.find('a word to the wise', '(w.)(..)')  -- two!
3       6       wo      rd
> = string.find('a word to the wise', '(w(..).)')  -- nested
3       6       word    or
```

You can see that `string.find()` returns the start and finish indices of the whole pattern and the *captures* that we made using brackets in the pattern.

This can be very useful in conjunction with the `string.gsub()` function, because as well as doing simple search and replace, we can process the capture we find before performing the replacement, e.g.,

```
> = string.gsub('a word to the wise', '(t%a+)', 'banana') -- simple replace
a word banana banana wise       2
>
> = string.gsub('a word to the wise', '(t%a+)',
>>    function(x) print(x); return string.upper(x) end)
to
the
a word TO THE wise      2
>
> = string.gsub('a word to the wise', '(w%l+)',
>>    function(x) return '_'..x..'_' end)
a _word_ to the _wise_  2
```

FindPage · RecentChanges · preferences
edit · history
Last edited April 6, 2008 12:22 am GMT (diff)