JavaScript

The API works as follows:

- `regex.exec( str )`
- `str.match( regex )`

These return all the capture groups defined in `regex` when matching `str`.

```
const regex = /^Price: (([€\$])(\d\d\.\d\d))$/;
const str = 'Price: €19.00';

const matches = regex.exec( str );
// str.match( regex ); // does the same

console.table( matches );
```

```
(index) Value
0       "Price: €19.00"
1       "€19.00"
2       "€"
3       "19.00"
index   0
input   "Price: €19.00"
```

As a side note, if you don't yet know `console.table`, check out my article on the console API.[1]

[1] www.zsoltnagy.eu/understanding-the-console-api-in-javascript-7-tips-for-smoother-debugging/

PHP

We will use `preg_match_all( $regex, $str, $result )`. PHP returns all the capture groups inside the `$result` array.

```
$regex = "/^Price: ((€|\$)(\d\d\.\d\d))$/";
$str = "Price: €19.00";

preg_match_all($regex, $str, $result);
print_r($result);
```

Execute the code, for example, in the PHP Sandbox.[2] The result is as follows:

```
Array
(
    [0] => Array
        (
            [0] => Price: €19.00
        )
    [1] => Array
        (
            [0] => €19.00
        )
    [2] => Array
        (
            [0] => €
        )
    [3] => Array
        (
            [0] => 19.00
        )
)
```

[2] http://sandbox.onlinephpfunctions.com/

Python

Let's recall the code we saw when we first tested regular expressions with Python, and let's place the correct regex and `test_str` values in there.

```
import re

regex = r"^Price: ((€|\$)(\d\d\.\d\d))$"

test_str = "Price: €19.00"

matches = re.finditer(regex, test_str)

for matchNum, match in enumerate(matches):
    matchNum = matchNum + 1

    print ("Match {matchNum} was found at {start}-{end}: {match}"
        .format(
            matchNum = matchNum,
            start = match.start(),
            end = match.end(),
            match = match.group()
        )
    )

    for groupNum in range(0, len(match.groups())):
        groupNum = groupNum + 1

        print ("Group {groupNum} found at {start}-{end}: {group}"
            .format(
                groupNum = groupNum,
                start = match.start(groupNum),
                end = match.end(groupNum),
                group = match.group(groupNum)
            )
        )
```

Once we paste this code in the Python shell,[3] the result becomes visible.

```
Match 1 was found at 0-13: Price: €19.00
Group 1 found at 7-13: €19.00
Group 2 found at 7-8: €
Group 3 found at 8-13: 19.00
```

As you can see, the capture groups are accessible via the `match` object.

Yes, this was the promised foreshadowing from the first introduction of the same Python code. Originally, the same code didn't find any capture groups in the expression, but now you can see that the inner `for` loop prints out the capture groups one by one.

Perl 5

We have seen that a simple match can be executed using the =~operator.

```
if ( "Price: €19.00" =~ /^Price: ((€|\$)(\d\d\.\d\d))$/ ) {
    print "Match";
}
```

We have already seen a few variables that provide us with some context to the match, such as the following:

```
if ( "Price: €19.00" =~ /^Price: ((€|\$)(\d\d\.\d\d))$/ ) {
    print "String before the match: ", $`, "\n";
    print "String after the match:", $', "\n";
    print "Matched string:", $&, "\n";
}
```

Similarly, there are variables defined for each capture group in the form of $1, $2, $3, and so on.

[3] https://www.python.org/shell/

```
if ( "Price: €19.00" =~ /^Price: ((€|\$)(\d\d\.\d\d))$/ ) {
    print "String before the match: ", $`, "\n";
    print "String after the match:", $', "\n";
    print "Matched string:", $&, "\n";
    print "First capture group:", $1, "\n";
    print "Second capture group:", $2, "\n";
    print "Third capture group:", $3, "\n";
}
```

The result of the execution is as follows:

```
String before the match:
String after the match:
Matched string:Price: €19.00
First capture group: €19.00
Second capture group: €
Third capture group:19.00
```

Reusing Captured Substrings Within a Regex

Suppose we are interested in extracting a price in between any positive number of highlighter characters. The following strings should all be matches:

```
****€19.00****
*-*€19.00*-*
--€19.00--
```

The following inputs are invalid:

```
€19.00     // no highlighter
*€19.00**  // length before and after don't match
**€19.00-- // character sequences don't match
++€19.00++ // not accepted characters
```