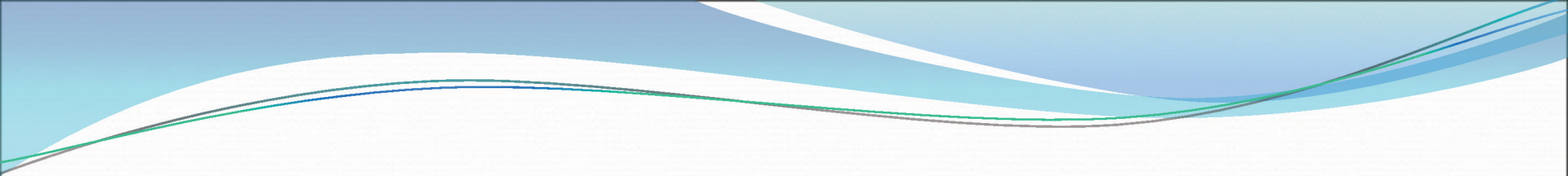


Regular Expression in PHP

I MSc CS

Sahaya Chithra .E

- 
- Tokenizing and parsing
 - Regular expression functions
 - Advanced array functions

Tokenizing and parsing

- The process of breaking up a long string into words is called Tokenizing.
- Php provides a special function for this purpose called `strtok()`.
- Syntax:
- `Strtok(string1, string2)`
- `String1` – string to be broken up
- `String2` – string containing all the *delimiters*.

Regular expression functions

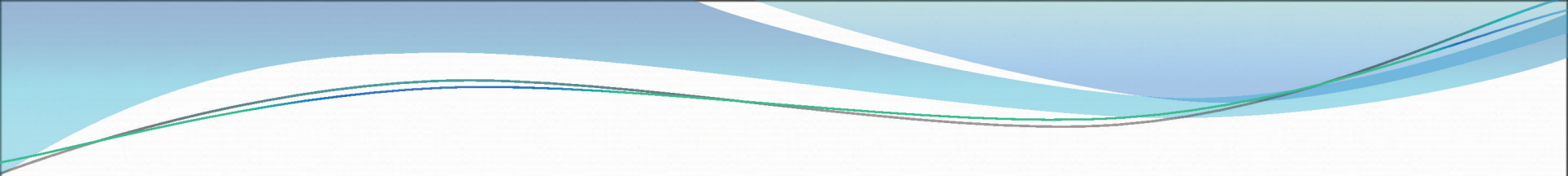
- **Regular expressions (or *regex*)** are patterns for string matching, with special wildcards that can match entire portions of the target string.
- There are two broad classes of regular expression
 - *POSIX* (extended) regex
 - *Perl-compatible regex*.

- POSIX-style regular expressions are ultimately descended from the regex pattern-matching machinery used in Unix command-line shells.
- Perl-compatible regex is a more direct imitation of regular expressions in Perl.

Rules for POSIX-style regular expressions

- Characters that are not *special* are matched literally. The letter *a* in a pattern, for example, matches the same letter in a target string.
- The special character `^` matches the beginning of a string only, and the special character `$` matches the end of a string only.
- The special character `.` matches any character.

- The special character `*` matches zero or more instances of the previous regular expression, and `+` matches one or more instances of the previous expression.
- A set of characters enclosed in square brackets matches any of those characters — the pattern `[ab]` matches either `a` or `b`. You can also specify a range of characters in brackets by using a hyphen — the pattern `[a-c]` matches `a`, `b`, or `c`.
- Special characters that are escaped with a backslash (`\`) lose their special meaning and are matched literally.

- 
- function **ereg(string1,string2)**, which takes as arguments a pattern string and a string to match against.
 - We can use an **ereg()** call to build a test function for our kind of web address.

- `<?php`
- `$a=array("hjhjh.6567@@@.ibm.com","WWW.java.sun.com",
"WWW.zend.com","WWW.IBM.COM","WWW.java.sun.com");`
- `function arr($s)`
- `{`
- `return(ereg('^WWW\\.[a-z]+\\.com$', $s));`
- `}`
- `while($t=array_pop($a))`
- `{`
- `if(arr($t))`
- `print("\"$t\" is a simple dot_com
");`
- `else`
- `print("\"$t\" is a not simple dot_com
");`
- `}`
- `?>`

Browser window showing the address bar with the URL `http://localhost/sara/regexp.php`. The browser title is `localhost`. The menu bar includes `File`, `Edit`, `View`, `Favorites`, `Tools`, and `Help`.

Search bar with `Secure Search` and `McAfee` icons. Navigation icons for home, back, forward, and search are visible on the right side of the search bar.

```
"WWW.java.sun.com" is a not simple dot_com  
"WWW.IBM.COM" is a not simple dot_com  
"WWW.zend.com" is a simple dot_com  
"WWW.java.sun.com" is a not simple dot_com  
"hjhjh.6567@@@.ibm.com" is a not simple dot_com
```

Windows taskbar showing icons for File Explorer, Microsoft Word, Internet Explorer, a green application icon, a red application icon, and a red application icon. The system tray on the right shows the date and time: `9:30 AM 2/17/2016`.

`^www\.[a-z]+\com$`

In this expression we have the ‘^’ symbol, which says that the www portion must start at the beginning of the string.

Then comes a dot (.), preceded by a backslash that says we really want a dot, not the special . wildcard character. Then we have a bracket-enclosed range of all the lowercase alphabetic letters.

The following + indicates that we are willing to match any number of these lowercase letters in a row, as long as we have at least one of them.

Then another literal ., the com, and the special \$ that says that com is the end of it.

POSIX Regular Expression Functions

1. `ereg(str1,str2)` - returns TRUE if the match was successful and FALSE otherwise.
2. `eregi()` - Identical to `ereg()`, except that letters in regular expressions are matched in a case-independent way.
3. `ereg_replace(str1,str2,str3)` - Takes three arguments: a POSIX regular expression pattern, a string to do replacement with, and a string to replace into. The function scans the third argument for portions that match the pattern and replaces them with the second argument. The modified string is returned.

4. `eregi_replace()` - Identical to `ereg_replace()`, except that letters in regular expressions are matched in a case-independent way.
5. `split()` - Takes a pattern, a target string, and an optional limit on the number of portions to split the string into. Returns an array of strings created by splitting the target string into chunks delimited by substrings that match the regular expression.
6. `spliti()` - Case-independent version of `split()`.

Perl-Compatible Regular Expressions

- The Perl compatible pattern:

`/pattern/`

- matches any string that has the string (or substring) pattern in it.

Common Perl-Compatible regular expression functions - overview

- **preg_match()** - Takes a regex pattern as first argument, a string to match against as second argument, and an optional array variable for returned matches. Returns 0 if no matches are found, and 1 if a match is found.
- **preg_match_all()** - Like **preg_match()**, except that it makes all possible successive matches of the pattern in the string, rather than just the first. The return value is the number of matches successfully made.
- **preg_split()** Takes a pattern as first argument and a string to match as second argument. Returns an array containing the string divided into substrings, split along boundary strings matching the pattern.

preg_replace() Takes a pattern, a replacement string, and a string to modify. Returns the result of replacing every matching portion of the modifiable string with the replacement string.

preg_replace_callback() - Like `preg_replace()`, except that the second argument is the name of a callback function, rather than a replacement string. This function should return the string that is to be used as a replacement.

preg_grep() - Takes a pattern and an array and returns an array of the elements of the input array that matched the pattern. Surviving values of the new array have the same keys as in the input array.

preg_quote() - A special-purpose function for inserting escape characters into strings that are intended for use as regex patterns. The only required argument is a string to escape; the return value is that string with every special regex character preceded by a backslash.