Influences on JavaScript

- **Java**: basic syntax, many type/method names
- **Scheme**: first-class functions, closures, dynamism
- **Self**: prototypal inheritance
- **Perl**: regular expressions

Historic note: *Perl* was a horribly flawed and very useful scripting language, based on Unix shell scripting and C, that helped lead to many other better languages.

- PHP, Python, Ruby, Lua, ...
- Perl was excellent for string/file/text processing because it built *regular expressions* directly into the language as a first-class data type. JavaScript wisely stole this idea.
What is a regular expression?

/\([a-zA-Z_\-]+@[([a-zA-Z_\-])+\.]+[a-zA-Z]{2,4}\)/

• regular expression ("regex"): describes a pattern of text
  - can test whether a string matches the expr's pattern
  - can use a regex to search/replace characters in a string
  - very powerful, but tough to read

• regular expressions occur in many places:
  - text editors (TextPad) allow regexes in search/replace
  - languages: JavaScript; Java Scanner, String split
  - Unix/Linux/Mac shell commands (grep, sed, find, etc.)
## String regexp methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.match(regexp)</code></td>
<td>returns first match for this string against the given regular expression; if global <code>/g</code> flag is used, returns array of all matches</td>
</tr>
<tr>
<td><code>.replace(regexp, text)</code></td>
<td>replaces first occurrence of the regular expression with the given text; if global <code>/g</code> flag is used, replaces all occurrences</td>
</tr>
<tr>
<td><code>.search(regexp)</code></td>
<td>returns first index where the given regular expression occurs</td>
</tr>
<tr>
<td><code>.split(delimiter[, limit])</code></td>
<td>breaks apart a string into an array of strings using the given regular as the delimiter; returns the array of tokens</td>
</tr>
</tbody>
</table>
Basic regexes

/abc/

• a regular expression literal in JS is written  /pattern/  
• the simplest regexes simply match a given substring

• the above regex matches any line containing "abc"
  ▪ **YES** : "abc", "abcdef", "defabc", ".=.abc.=."
  ▪ **NO** : "fedcba", "ab c", "AbC", "Bash",...
Wildcards and anchors

- (a dot) matches any character except \n
  - /.oo.y/ matches "Doocy", "goofy", "LooPy", ...
  - use \. to literally match a dot . character

^ matches the beginning of a line; $ the end
  - /^if$/ matches lines that consist entirely of if

\< demands that pattern is the beginning of a word;
\> demands that pattern is the end of a word
  - /\<for\>/ matches lines that contain the word "for"
String match

`string.match(regex)`

- if string fits pattern, returns matching text; else `null`
  - can be used as a Boolean truthy/falsey test:
    ```javascript
    if (name.match(/^[a-z]+$/)) { ... }
    ```

- `g` after regex for array of *global* matches
  - "obama".match(/.a/g) returns ["ba", "ma"]

- `i` after regex for case-*insensitive* match
  - `name.match(/Marty/i)` matches "marty", "MaRtY"
String replace

\[ \text{string}.\text{replace}(\text{regex}, \ "text") \]

- replaces first occurrence of pattern with the given text
  - var state = "Mississippi";
    state.replace(/s/, "x") returns "Mixissippi"

- \text{g} after regex to replace all occurrences
  - state.replace(/s/g, "x") returns "Mi**x**i**x**i**x**ippi"

- \text{return}s the modified string as its result; must be stored
  - state = state.replace(/s/g, "x");
Special characters

| means OR
- `/abc|def|g/` matches lines with "abc", "def", or "g"
- precedence: `^Subject|Date:` vs. `^(Subject|Date):`
- There's no AND & symbol. Why not?

() are for grouping
- `/^(Homer|Marge) Simpson/` matches lines containing "Homer Simpson" or "Marge Simpson"

\ starts an escape sequence
- many characters must be escaped: `/\$[^.\[\]()]^*+?`
- "\n" matches lines containing ".\n"
Quantifiers:  *  +  ?

*  means 0 or more occurrences
  - /abc*/ matches "ab", "abc", "abcc", "abccc", ...
  - /a(bc)/ matches "a", "abc", "abcbc", "abcbcbc", ...
  - /a.*a/ matches "aa", "aba", "a8qa", "a!?_a", ...

+  means 1 or more occurrences
  - /a(bc)+/ matches "abc", "abcbc", "abcbcbc", ...
  - /Goo+gle/ matches "Google", "Gooogle", "Gooooogle", ...

?  means 0 or 1 occurrences
  - /Martina?/ matches lines with "Martin" or "Martina"
  - /Dan(iel)?/ matches lines with "Dan" or "Daniel"
More quantifiers

\{\text{min}, \text{max}\} \text{ means between } \text{min} \text{ and } \text{max} \text{ occurrences}

- /a(bc){2,4}/ matches lines that contain "abcbcb", "abcbcbcb", or "abcbcbcbcb"

- \text{min} \text{ or } \text{max} \text{ may be omitted to specify any number}

- \{2,\} \quad 2 \text{ or more}
- \{,6\} \quad \text{up to 6}
- \{3\} \quad \text{exactly 3}
[ ] group characters into a *character set*; will match any single character from the set

- `/[bcd]art/` matches lines with "bart", "cart", and "dart"
- equivalent to `/\(b|c|d\)art/` but shorter

• inside [], most modifier keys act as normal characters
  
  - `/what[.!*?]*/` matches "what", "what.", "what!", "what?**!", ...

  – *Exercise*: Match letter grades e.g. A+, B-, D.
Character ranges

• inside a character set, specify a range of chars with -
  ▪ /[a-z]/ matches any lowercase letter
  ▪ /[a-zA-Z0-9]/ matches any letter or digit

• an initial ^ inside a character set negates it
  ▪ /[^abcd]/ matches any character but a, b, c, or d

• inside a character set, - must be escaped to be matched
  ▪ /\[-+]?[0-9]+/ matches optional - or +, followed by at least one digit

  – Exercise: Match phone numbers, e.g. 206-685-2181.
Built-in character ranges

- \b  word boundary (e.g. spaces between words)
- \B  non-word boundary
- \d  any digit; equivalent to [0-9]
- \D  any non-digit; equivalent to [^0-9]
- \s  any whitespace character; [\f\n\r\t\v...]
- \S  any non-whitespace character
- \w  any word character; [A-Za-z0-9_]
- \W  any non-word character
- \xhh, \uhhhh  the given hex/Unicode character

- /\w+\s+\w+/  matches two space-separated words
Regex flags

/pattern/g  global; match/replace all occurrences
/pattern/i  case-insensitive
/pattern/m  multi-line mode
/pattern/y  "sticky" search, starts from a given index

• flags can be combined:
  /abc/gi matches all occurrences of abc, AbC, aBc, ABC, ...
Back-references

- text "captured" in ( ) is given an internal number; use \textit{number} to refer to it elsewhere in the pattern
  - \textbackslash{} is the overall pattern,
  - \textbackslash{}1 is the first parenthetical capture, \textbackslash{}2 the second, ...
  - Example: "A" surrounded by same character: /\( . \)A\textbackslash{}1/

  - variations
    - (?:\textit{text}) match \textit{text} but don't capture
    - a(?=b) capture pattern \textit{b} but only if preceded by \textit{a}
    - a(?!b) capture pattern \textit{b} but only if not preceded by \textit{a}
Replacing with back-references

- you can use back-references when replacing text:
  - refer to captures as $number in the replacement string
  - Example: to swap a last name with a first name:

    ```javascript
    var name = "Durden, Tyler";
    name = name.replace(/(\w+),\s+(\w+)/, "$2 $1");
    // "Tyler Durden"
    ```

- Exercise: Reformat phone numbers from 206-685-2181 format to (206) 685.2181 format.
The RegExp object

```javascript
new RegExp(string)
new RegExp(string, flags)
```

- constructs a regex dynamically based on a given string
  ```javascript
  var r = /ab+c/gi;  // is equivalent to
  var r = new RegExp("ab+c", "gi");
  ```

  - useful when you don't know regex's pattern until runtime
    - Example: Prompt user for his/her name, then search for it.
    - Example: The empty regex (think about it).
Working with RegExp

• in a regex literal, forward slashes must be \ escaped:
  `/http[s]?://\w+.com/`

• in a new RegExp object, the pattern is a string, so the usual escapes are necessary (quotes, backslashes, etc.):
  `new RegExp("http[s]?://\w+.com")`

• a RegExp object has various properties/methods:
  - properties: global, ignoreCase, lastIndex, multiline, source, sticky; methods: exec, test
Many editors allow regexes in their Find/Replace feature.

```
grep -e "[pP]hone.*206[0-9]{7}" contacts.txt
```