String & Regular Expression Language Reference

$s_i^j \triangleq s_i s_{i+1} \ldots s_{j-1}$	PHP	Ruby	Javascript	Python	Java
delimiter	' (no escape chars or embedded variables) or "	' or " / for regexps	' or " for strings / for regexps	' or "	"
type	string q, r, s, t array[string] a, m	String q, s, t Regexp r MatchData m	String q, s RegExp|String r	str q, s, t *RegexObj* r *MatchObj* m	String q, s, t Pattern r Matcher m
converter		obj.to_s	obj.toString()	str(obj)	obj.toString()
copy			**new** String(s)	s[:]	**new** String(s)
equivalence	s==q	s==q	s==q	s==q	s.equals(q)
length $\|s\|$	strlen(s)	s.length	s.length	len(s)	s.length()
index/slice					
s_i	s[i]	s[i,1] $i \in [-\|s\|, \|s\|]$, else **nil**	s[i]; s.charAt(i) $i \in [0, \|s\|)$	s[i] $i \in [-\|s\|, \|s\|)$, *else error*	s[i]; s.charAt(i) $i \in [0, \|s\|)$, *else error*
s_i^j	substr(s,i,j-i) $i \in [-\|s\|, \|s\|), j - i > 0$ substr(s,i,j) $i \in [-\|s\|, \|s\|), j \in [-\|s\|, 0)$	s[i..j-1] $i,j \in [-\|s\|, \|s\|)$	s.slice(i,j) $i,j \in [-\|s\|, \|s\|)$ s.substring(i,j) $i \in [0, \|s\|), j \in (i, \|s\|]$	s[i:j] $i,j \in [-\|s\|, \|s\|)$	s.substring(i,j) $i \in [0, \|s\|], j \in [i, \|s\|]$, *else error*
s_i^{i+L}	substr(s,i,L) $i \in [-\|s\|, \|s\|), L > 0$	s[i,L] $i \in [-\|s\|, \|s\|], L > 0$, else **nil**	s.substr(i,L) $i \in [-\|s\|, \|s\|), L \in (0, \|s\|)$	s[i:i+L] $i \in [-\|s\|, \|s\|), L > 0$	s.substring(i,i+L) $i \in [0, \|s\|], i + L \in [i, \|s\|]$, *else error*
$s_i^{\|s\|}$	substr(s,i)	s[i..-1] $i \in [-\|s\|, \|s\|]$, else **nil**	s.slice(i) $i \in [-\|s\|, \|s\|)$	s[i:] $i \in [-\|s\|, \|s\|)$	s.substring(i) $i \in [0, \|s\|]$, *else error*
character case	strtolower(s) strtoupper(s) ucfirst(s) ucwords(s)	s.downcase! s.upcase! s.capitalize!	s.toLowerCase() s.toUpperCase()	s.lower() s.upper() s.capitalize() s.title()	s.toLowerCase() s.toUpperCase()
trim	trim(s)	s.strip!		s.strip()	s.trim()
split & join	explode(q,s) preg_split(r,s) join(q,a)	s.split(q) s.split(r) s.split; s.split(' ') *by whitespace chunks*	s.split(q) s.split(r)	s.split(q); re.split(q,s) s.split() *by whitespace chunks* s.splitlines() q.join(…)	s.split(q) *q is a regexp pattern string*
containment	strpos(s,q)!==**FALSE**	s.include?(q)	s.indexOf(q)>-1	q **in** s	s.contains(q)
startswith				s.startswith(q)	s.startsWith(q)
endswith				s.endswith(q)	s.endsWith(q)
char code at	ord(s[i])	s[i]	s.charCodeAt(i)	ord(s[i])	s.codePointAt(i)
code2char	chr(i)	i.chr	String .fromCharCode(i)	chr(i)	**new** Character((**char**)i) .toString()
int2string	strval(i)	i.to_s	**new** String(i)	str(i)	Integer.toString(i)
string2int	intval(s)	s.to_i	**new** Number(s)	int(s)	Integer.parseInt(s)

String & Regular Expression Language Reference

$s_i^j \triangleq s_i s_{i+1} \ldots s_{j-1}$	PHP	Ruby	Javascript	Python	Java
search	`str_rpos(s,q)` *FALSE on failure*	`s.rindex(q)` *nil on failure* `s=~r; s.rindex(r)` *nil on failure*	`s.lastindexOf(q)` *-1 on failure* `s.search(r)` *-1 on failure*	`s.rfind(q)` *-1 on failure* `s.rindex(q)` *ValueError on failure*	`s.lastindexOf(q)` *-1 on failure*
match	`preg_match(q,s)` *returns whether there is a match: 0 (failure) or 1, or FALSE on error* `preg_match(q,s,&m)` *m is the destination array for matched strings, indexed by group* `preg_match_all(q,s, &m,f)` *returns the number of matches, and stores the matches in m structured according to the provided flag f, which is one of:* PREG_PATTERN_ORDER *indexed as m[i][j], where i is the group number and j is the match number* PREG_SET_ORDER *indexed as m[j][i], where i is the group number and j is the match number*	`r=Regexp.compile(q)` `s[r]` *the entire first match* `s[r,i]` *the i^th group* `m=s.match(r);` `m=s.match(s)` *match result; nil on failure* `m[i]` *substring matched by the i^th group (nil if that group is unused* in m)* `m.captures` *group list* `m.begin(i)` `m.end(i)` *start and end offsets of the substring matched by the i^th group (nil if group unused* in m)* `$~` *the MatchData object from the most recent match* `$&` *the entire text of the most recent match* `$1, $2 …` *text matched by the 1st, 2nd, … groups in the most recent match*	`s.search(r)` *-1 on failure* `r.test(s)` *tests for a (next) match* `s.match(r)` *match result; null on failure* `s.match(r)[i]` *the i^th matched group, if r is local, or the i^th matched substring if r is global* `s.match(r).index` *match offset, if r is local* `r.exec(s)` *result for r's first match in s, if r is local, or its next match in s, if r is global; null if no (more) matches* `r.exec(s)[i]` *the i^th matched group in the result* `r.exec(s).index` *offset of the match* `RegExp.$&` *the entire text of the most recent match* `RegExp.$1, RegExp.$2 …` *text matched by the 1st, 2nd, … groups in the most recent match*	`r=re.compile(q)` `m=re.search(q,s);` `m=r.search(s,i=0)` *the first match of the pattern at or after position i (default: 0) in s. None on failure.* `m=re.match(q,s);` `m=r.match(s,i=0)` *like search, but only finds matches starting at position i* `m.group(i=0)` *substring matched by the i^th group (None if that group is unused* in m)* `m.groups()` *group list* `m.start(i=0)` `m.end(i=0)` *start and end offsets of the substring matched by the i^th group (-1 if group unused* in m)* `re.findall(q,s);` `r.findall(s)` *list of matched groups from all matches in s, with '' for groups unused in a particular match. Entries are g-tuples if the pattern contains g>1 groups, or entire matches if g=0.*	`r=Pattern.compile(q)` `m=r.matcher(s)` `s.matches(q);` `m.matches()` *checks for a match* `m.find()` *advances to the next match, returning true if a next match was found and false otherwise* `m.group(i=0)` *substring matched by the i^th group (null if that group is unused* in m)* `m.start(i=0)` `m.end(i=0)` *start and end offsets of the substring matched by the i^th group (-1 if group unused* in m)*
replace	`str_replace(q,t,s)` `preg_replace(r,t,s)` `str_replace(q,t,s,1)` `preg_replace(r,t,s,1)`	`s.gsub!(q,t)` `s.gsub!(r,t)` `s.sub!(q,t)` `s.sub!(r,t)`	`s.replace(r,q)`	`s.replace(q,t)` `re.sub(q,t,s)` `s.replace(q,t,1)` `re.sub(q,t,s,1)`	`s.replaceAll(q,t)` `m.replaceAll(t)` `s.replaceFirst(r,q)` `m.replaceFirst(t)`
format	`sprintf(s,…)`	`s % […]`		`s % (…)` `s.format(…)`	`String.format(s, …)`

***Regular expressions note:** An expression or group that matches an empty string nevertheless qualifies as *used*. A group may be *unused* in a match if it is part of a disjunction, for example.

String & Regular Expression Language Reference

$s_i^j \triangleq s_i s_{i+1} \ldots s_{j-1}$	PHP	Ruby	Javascript	Python	Java
more on literals	Multiline string literals include heredocs (like double-quoted strings): `$s = <<<EOF` … `EOF;` PHP5 also has nowdocs (like single-quoted strings): `$s = <<<'EOF'` … `EOF;` The heredoc/nowdoc terminator must not be indented.		`/…/g` or `RegExp(…, "g")` for a regexp that does global matching	String literals: `u` prefix for Unicode strings (Python 2) `r` prefix for *raw strings* (backslashes as literals) *triple-quoted strings* (surrounded by `'''`) can span multiple lines	
other				`s.islower()` `s.isnumeric()` `s.isdigit()` `s.isalpha()` `s.isalnum()` `s.isspace()` `s.isupper()`	

http://www.php.net/manual/en/ref.strings.php

http://www.ruby-doc.org/core/classes/String.html

http://www.devguru.com/technologies/javascript/10792.asp ; http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

http://docs.python.org/3.0/library/stdtypes.html#sequence-types-str-bytes-bytearray-list-tuple-range

http://java.sun.com/javase/6/docs/api/java/lang/String.html

Compiled by Nathan Schneider ▫ **nathan.cl**

Last updated 13 May 2011