Ruby – Methods

General Operations

|  |  |
| --- | --- |
| **Call** | **Description** |
| print *expression* | Prints expression on first line of output |
| puts *expression* | Prints and automatically adds new line at end of expression, returns ‘nil’ if used in method |
| p *expression* | Prints raw version of expression and if used in method it will return the value of the object |
| gets | Asks user for input |

Methods

|  |  |
| --- | --- |
| *string1* <=> *string2* | combined comparison operator returns 1 if string 1 is greater than string 2, -1 if vise versa, and 0 if both are equal. Can be used on other objects to sort. |
| .length | returns number of elements in string or array |
| *string.*count “c1” “c2” | returns number of characters counted between the matched strings c1 and c2. Strings can be excluded using ^, ranges of letter can be used using -. |
| *array.*count (val) or {|x| *boolean*} | returns number of elements in array. If optional val is given, will return number of elements matching val. If block is given, will return number matching boolean |
| .upcase (!) | returns string in upcase. Addition of ! mutates object but returns nil if nothing has changed |
| .downcase (!) | returns string in downcase. Addition of ! mutates object but returns nil if nothing has changed |
| .reverse (!) | returns string or array in reverse. Addition of ! mutates object. |
| .capitalize (!) | returns string in with first letter capitalized. Addition of ! mutates object but returns nil if nothing has changed |
| .chomp(arg) (!) | returns string with record separator (arg blank) or specified argument removed. Addition of ! mutates object |
| .sub(/[regex]/, “val”) | substituted first match of the value or regex with value of val. If val is a hash and the matched text one of its keys then the value substituted will be the hash key value. This can also be done by creating a regex.union of the (hash.keys), and inputting this as the first argument. |
| .gsub(/[regex]/, “val”) | similar to .sub however replaces all matched values. since gsub uses regex it is possible to use the capture groups in the sub value if they are defined in the regex. Capture group can be simply inserted in the <val> string by using ‘\n’ where n is the capture group number. |
| .next or .succ | will return next integer or string |
| .cover? | returns true is object is inbetween range |
| *array*.each { |n| *expression* }  *hash*.each { |key, value| *expression*}  *hash*.each\_key { |key| *expression*}  *hash*.each\_value { |value| *expression*} | For array: for each element in the array, each runs the block, passing it the element as an argument, however does not collect the results as array  For hash: for entries in the hash, each runs the block, passing it the key/value pair as an argument. This can be simplified using \_key or \_value after .each. |
| *.each\_with\_index {|val, index| exp }* | Each over an array with acululating index value |
| *array*.map { |n| *expression* }  *hash*.map { |k,v| [k, v.to\_sym] }.to\_h | returns array with each element containing the result of the original array element run passed through the block . Can be used on hash, and return a hash using .to\_hash |
| *array*.select { |n| *boolean*? }  *hash*.select { |key, value| *Boolean?*} | For array: returns array of values for which Boolean expression was true  For hash: returns entries which match boolean |
| *Hash.*sort\_by {|key, value| value} | Returns hash as array sorted by either value or key |
| *integer*.times{ *expression* } | repeat expression integer times |
| .split(*delimiter, limit*) | Splits a string into an array based on the delimiter given. Delimiter can be string “*string*” or a regex “/regex/”, the string will be split each time delimiter is matched. If limit argument is added, the string will be split into max limit parts. |
| *array*.join(‘*separator’*) | returns joined array with optional separator string between elements |
| *Array*.sort {|a, b| exp i.e. a <=> b} | Sorts array via <=> expression of a and b values in array, expression can be complex as long as values are computable |
| *Array.*delete(val) | Deletes all instances of value from array |
| array.sort\_by { |val| *expression*} | returns array sorted by expression on val, such as ‘a[1] would sort by the second value of a nested array |
| *hash*.delete*(key*) | deletes specific hash key (via mutation) and returns the value pair, if no key is found it returns nil |
| *.object\_id* | returns object id of object |
| *array.*push(val) | pushes given object (val) to end of array as new element then returns new array |
| .to\_sym | converts string to symbol |
| .to\_s | convert object to string |
| .to\_i | convert object to integer |
| .to\_h | converts a two element 3D array to a hash |
| .to\_a | converts range, string or integer to array |
| .to\_s(n) | converts integer to its corresponding binary number (with base n) returning a string |
| .to\_i(n) | converts binary number (with base n) in string form to corresponding integer |
| .upto(*val*) {|n| *expression*} | iterates through successive values, performing optional expression. can be linked with .to\_a to produce array of iterations |
| .downto(*val*) {|n| *expression*} | similar to .upto however iterates down |
| .respond\_to?(*method*) | returns true if object responds to method |
| .all? {|n| boolean} | returns true if all elements in array or range are true when passed into Boolean expression |
| *string*.ord | returns integer ACSII code of first letter of string |
| *string*.bytes | returns integer ACSII code of each letter of string, in an array. (shorthand for .each\_byte.a |
| *integer.*chr | returns integer ACSII code convereted to string, or other encoded number if encoding is specified |
| *.sum* | returns sum of all elements in array |
| .detect {|n| *Boolean*} | returns first result which is not false, can be used on hash of values to find match or closest value. |
| *string*.chr | returns first character of string |
| *array.*slice(<index1>,<index2>) | returns values of array from index1 up to but not including index2 |
| *array.*pop | removes last value of an array |
| *array*.delete\_at(<index>) | deletes the index of the array specified |
| *array.*include?(<value>) | will return Boolean if value is included or not |
| *array*.delete(<value>) | Deletes all instances of value in array |
| *array*.shift | Deletes first element of array and moves all down |
| *Array.*delete\_if {|n| exp} | Deletes array elements if they are false |
| *Array.reject {|n| exp|* | Returns all elements which are false |
| .inject(0) { |sum, n| sum + n } | Will provide accumulator for range or array starting at value 0 or other specified, will also do any other expression given, where the sum value will be remembered and n the next item. |
| *array*.shift(n) | Removes the first n elements of an array and returns them. Array is modified and items removed are returned. |
| *string*.tr(“<string1>”, “<string2>” | Will transpose any element of string matched in <string1> to the element at the corresponding index of <string2>, returns string with any matched element changed. |
| .abs | Returns absolute value of integer or float |
| *string*.match(/<regex>/, n)<matched-data-parameter> | Returns matched data object if matched, optional parameter defines which index of the string to begin search on, optional matched data parameter (e.g. [0] will return the string matched instead of matched data object). Optional ? after match to turn result into boolean |
| *Matched-data-object.*[n] | * [0] – returns all matched charaters * [1, 2] – returns matches one and two * [i, length] – returns matches from index for length in array * [n1..n2] – returns matches n1 to n2 in array |
| *String.*scan(/regex/) {|match| exp} | For each match found, the match is either passed into a results array and returned, or passed into the optional block. |
| *String.*center(n, ‘str’)  *String.ljust*(n, ‘str’)  *String.rjust*(n, ‘str’) | Justify string in string length n, pad out any extra blank space with optional string argument |
| *String.empty?* | Returns true is string has no charaters |
| *String.*capitalize | Capitalizes first letter of string, use ! to mutate. |
| *String.*strip | Removes white space from string, similar to .chomp |
| *<hash1>.merge(<hash2)*  *{|key, val1, val2| exp}* | Merges hash two into hash one overwritting exisiting values, if keys do not exist new ones are made. Optional block allows for control over overwritting. |
| *Array.*any? {|s| exp} | Returns true if any object (s) in array matches expression |
| *Array*.find {|n| /regex/ =~ n} | Iterates over array and returns match to regex or expression. |
| *Array.find\_index{|n| exp}* | Iterates over array and returns first index which matches expression |
| *Hash.invert* | Returns hash with keys and values switched |
| *Hash.*flatten | Returns hash keys and values flattened into an array |
| *.*freeze | Makes an object immuttable, returns error if anyone trys to mutate |
| *Array.*rotate(index) | Rotates array at index, so index will become beginning of array and any values before will put at the end of the array. |
| *Hash.*fetch(key, <else>) | Return value of key if found in hash, or return <else>, <else> can be a block of code or if ommited it will return an KeyError |
| *Hash.fetch\_values(<key1>, <key2>) <{|k| block}>* | Returns array with values from matched keys, if any keys are not found the code will return error or return the optional block of code executed on the missing keys |
| *Hash.*has\_key?(key) | Returns boolean if keys is in hash or not |
| *Hash.*value?(value) | Returns boolean if value is in hash or not |
| *String*.chop | Removes last value of string |