```racket
#lang racket
(require rackunit)

(define (join-all-words lst sep)
  (string-join (filter (lambda (x)(string? x)) lst) sep))

(join-all-words (list 1 4 "cat" #f "walks" 4) " ")

(define (lst-alphabetized? lst)
  (cond ((< (length lst) 2) #t)
        ((string<=? (first lst)
                    (first (rest lst)))
             (lst-alphabetized? (rest lst)))
        (else #f)))

;-----------------------------------------
; Test cases
```

```
;
----------------------------------------
----------------------

(define numbers (list 1 2 3 ))

(check-true (string?
(join-all-words (list 1 2
"cat") ""))
            "Failed string
test")
(check-false (number?
(join-all-words (list 1 2
"cat") ""))
            "Failed string
test")

(check-equal? (join-all-words
numbers "")
              ""
              "Failed
no-string case")

(check-equal? (join-all-words
(list ) "")
```

```
                ""
              "Failed empty list
case")

;(check-equal? (join-all-words
5 "")
              ; (void )
              ; "Failed
non-list case")

(check-true (lst-alphabetized?
(list ))
              "Failed empty list
case")

;(check-true
(lst-alphabetized? (list 1 2
3))
;              "Failed
non-string case")

(check-true (lst-alphabetized?
(list "at" "bat" "cat"))
              "Failed
alphabetized case")
```

```
47
48 (check-false
48 (lst-alphabetized? (list "bat"
48 "at" "cat"))
49                  "Failed
49 non-alphabetized case")
50
51 (check-false
51 (lst-alphabetized? (list 1))
52                 "Failed singleton
52 non-string case")
53
54 (check-false
54 (lst-alphabetized? (list "at"
54 "abc"))
55                  "Failed
55 same-letter string case")
56
57
58
```