

```

1 #lang racket
2 (require rackunit)
3
4 (define (join-all-words lst
4 sep)
5   (string-join (filter (lambda
5 (x)(string? x)) lst) sep))
6
7 (join-all-words (list 1 4
7 "cat" #f "walks" 4) " ")
8
9 (define (lst-alphabetized? lst)
10   (cond ((< (length lst) 2) #t)
11         ((string=? (first
11 lst)
12                    (first
12 (rest lst))))
13         (lst-alphabetized?
13 (rest lst)))
14         (else #f)))
15
16 ;
16 -----
16 -----
17 ; Test cases

```

```
18 ;
18 -----
18 -----
19
20 (define numbers (list 1 2 3 ))
21
22 (check-true (string?
22 (join-all-words (list 1 2
22 "cat") ""))
23             "Failed string
23 test")
24 (check-false (number?
24 (join-all-words (list 1 2
24 "cat") ""))
25             "Failed string
25 test")
26
27 (check-equal? (join-all-words
27 numbers ""))
28             ""
29             "Failed
29 no-string case")
30
31 (check-equal? (join-all-words
31 (list ) ""))
```

```
32         ""
33         "Failed empty list
33 case")
34
35 ;(check-equal? (join-all-words
35 5 ""))
36         ; (void )
37         ; "Failed
37 non-list case")
38
39 (check-true (lst-alphabetized?
39 (list )))
40         "Failed empty list
40 case")
41
42 ;(check-true
42 (lst-alphabetized? (list 1 2
42 3)))
43 ;         "Failed
43 non-string case")
44
45 (check-true (lst-alphabetized?
45 (list "at" "bat" "cat")))
46         "Failed
46 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
```