The function `reverse` produces its argument but with the argument's elements in reverse order.

Beside each of the following twelve expressions, write down its value or the word "error" if it produces an error ...

```
(first  "ant"  "bee"  "cat"  "dog")   (first  (list  "ant"  "bee"))
(rest  "ant"  "bee"  "cat"  "dog")   (rest  (list  "ant"  "bee"))
(reverse  "ant"  "bee"  "cat"  "dog")  (reverse  (list  "ant"  "bee"))

(first  (list))  (first  (list  "ant"))
(rest  (list))  (rest  (list  "ant"))
(reverse  (list))  (reverse  (list  "ant"))
```

The function `empty?` determines whether its argument has no elements, in other words whether the length of the list is zero.

Beside each of the following four expressions, write down its value, or the word "error" if it produces an error ...

```
(empty?)  (empty?  (list  "nt"))  (empty?  (list))  (empty?  "nt")
```

The functions `first` and `rest` require a non-empty list argument.

For each of the following, write down the steps ...

```
(step  (first  (list  (text-join  "a"  "nt")  (text->image  "bee")  (+ 1 2))))
```

```
(step  (rest  (list  (text-join  "a"  "nt")  (text->image  "bee")  (+ 1 2))))
```

The function `reverse` produces its argument but with the argument's elements in reverse order. For the following, write down the steps ...

```
(step  (first  (reverse  (rest  (reverse  (list  (+ 50 7)  (filled-square 10)  (text-join  "e"  "mu")))))))
```
★ Question 2.
For each of the following function definitions, IF IT IS GRAMMATICALLY CORRECT THEN write down the values of the two expressions following it, and write down the steps for the corresponding expression that calls it, including any error messages during those steps.

(define (f.1 10)
  (oval 20 10))
(unary? f.1) (binary? f.1) (step (f.1 20))

(define (f.2 x)
  (oval 20 10))
(unary? f.2) (binary? f.2) (step (f.2 "1000"))

(define (f.3 "text")
  (text-join "text" "!"))
(unary? f.3) (binary? f.3) (step (f.3 "text"))

(define (f.4 (list a b))
  (rotate (triangle a b)))
(unary? f.4) (binary? f.4) (step (f.4 (list 10 15)))

(define (f.5 (list a b))
  (map circle (list a b)))
(unary? f.5) (binary? f.5) (step (f.5 10 15))

(define (f.6 a b)
  (map circle a b))
(unary? f.6) (binary? f.6) (step (f.6 10 15))

(define (f.7 a b)
  (map circle (list b a)))
(unary? f.7) (binary? f.7) (step (f.7 10 15))

(define (f.8 a)
  (map circle a))
(unary? f.8) (binary? f.8) (step (f.8 (list 10 15)))

(define (f.9 a)
  (map circle (list a)))
(unary? f.9) (binary? f.9) (step (f.9 10 20))

(define (f.10 a b)
  scale a b)
(unary? f.10) (binary? f.10) (step (f.10 (star 10) 20))

(define (f.11 amy)
  (text-join "amy" "("))
(unary? f.11) (binary? f.11) (step (f.11 "clara"))

(define (f.12 amy)
  (text-join amy "("))
(unary? f.12) (binary? f.12) (step (f.12 "clara"))

(define (f.13 amy)
  amy "amy")
(unary? f.13) (binary? f.13) (step (f.13 "clara"))

(define (f.14 amy)
  (text-join amy "amy"))
(unary? f.14) (binary? f.14) (step (f.14 "clara"))

(define (f.15 amy)
  (list amy "amy"))
(unary? f.15) (binary? f.15) (step (f.15 "clara"))
;;; Question 3.
;;; The function zero? takes a number and determines whether it's zero. The function positive? takes a number and determines whether it's positive.
;;; Beside each of the following two expressions, write down its value, or the word "error" if it produces an error ...

(map unary? (list zero? positive?))  (map binary? (list zero? positive?))

; Assume the following definition:
(define x -5)

; Write out the following steps ... 

(step (map zero? (list x (dec 1) 4)))  (step (map positive? (list -12 (- x) (- 12 12) (- 1 2) (- 2 1))))

;;; Question 4.
;;; The function text->list takes a text and produces a list containing each of its characters. Beside each of these two expressions write down its value.

(unary? text->list)  (binary? text->list)

; Assume the following definition:
(define k "cat")

; For each function call below, IF IT HAS EXACTLY ONE ARGUMENT EXPRESSION, then fill in the comment beside it by writing down the type of the argument.

; (function, boolean, list, image, number, or text), OTHERWISE don't fill in the comment. Then write down the value of the whole function call below it, or the word "error" if it produces an error.

((text->list "ok") ; type of the argument: **********
 (text->list k) ; type of the argument: **********

(text->list "o" "k") ; type of the argument: **********
 (text->list "k") ; type of the argument: **********

(text->list (list "o" "k")) ; type of the argument: **********
 (text->list list doge)) ; type of the argument: **********

(text->list) ; type of the argument: **********
 (text->list (list "d" "o" "g" "e")) ; type of the argument: **********

(text->list "") ; type of the argument: **********
 (text->list (list "d" "o" "g" "e")) ; type of the argument: **********

; Write out the following steps ...

(step (same? (length (text->list k)) (text-length k)))

(step (same? (length (text->list "!")) (text-length "!")))

(step (same? (length (text->list "kitty")) (text-length "kitty")))

(step (same? (length (text->list ")")) (text-length ")))