```
#lang racket

(define x 5)

(match x
  (5 "five")
  (10 "ten")
  (else "other"))

(define (match-numbers y)
  (match y
    (1 5)
    (2 10)
    (2 5)
    (else 10)))

(match-numbers 2)
(match-numbers 4)


(define (match-type y)
  (match y
    ((? string? y) "I'm a string!")
    ((? number? y) "I'm a number!")
    ((? boolean? y) "I'm a boolean!")
    ((? list? y) "I'm a list!")
    (_ "I'm something else!")))

(match-type 5)
(match-type "hi")
(match-type #\c)

(define (get-last lst)
  (match lst
    ((list a) a)
    ((list a b) b)
    ((list a b c) c)))

(get-last (list 4 5 6))
(get-last (list 2))

(define (get-last-2 lst)
  (match lst
    ((list a) a)
    ((list a b) b)
    ((list _ _ c) c)))

(get-last-2 (list 1 2 3))

(define tenlist (list 1 2 3 4 5 6 7 8 9 10))

```

```
(match tenlist
  ((list 1) "length 1")
  ((list n ... 10) "last item is 10"))

(define (lotsa-strings str)
  (match str
    ((list _ (? string? x) ... y ) (append x (list y)))
    (_ void)))

(lotsa-strings (list "cats" "love" "pickles"))
(lotsa-strings (list "pickles"))
(lotsa-strings (list "cats" "love" 5 "pickles"))

(define (get-last-3 lst)
  (match lst
    ((list _ ... a) a)))

(get-last-3 (list 1))
(get-last-3 (list 1 2 3 4))
(get-last-3 (list 1 2))
;(get-last-3 (list ))

(define (palindrome-checker lst)
  (match lst
    ((list a) #t)
    ((list a a) #t)
    ((list a b a) #t)
    ((list a b b a) #t)
    ((list a b c b a) #t)
    (else #f)))

(palindrome-checker (list 1))
(palindrome-checker (list 1 2))
(palindrome-checker (list 1 2 3 2 1))

(define (dups lst)
  (match lst
    ((list (? string? x)  _ ... x  _ ...) #t)
    (else #f)))

(dups (list "cat" "eats" "pickles"))
(dups (list "cat" "eats" "cat" "pickles"))
(dups (list "cat" "cat"))

(define (string-multiply str rep)
  (letrec ((helper (lambda (s n res)
                     (if (= n 0)
                         res
                         (helper s (- n 1) (string-append res s))))))
    (helper str rep "")))

```

```
(string-multiply "cat" 5)

(expt 5 2)

(define (generic-power x n)
  (match x
    ((? string? y)(string-multiply y n))
    ((? number? y)(expt y n))
    (_ (printf "ERROR: WRONG DATA"))))

(generic-power 5 2)
(generic-power "cat" 2)
(generic-power #t 2)
```