

```
1 #lang racket
2
3 (define x 5)
4
5 (match x
6   (5 "five")
7   (10 "ten")
8   (else "other"))
9
10 (define (match-numbers y)
11   (match y
12     (1 5)
13     (2 10)
14     (2 5)
15     (else 10)))
16
17 (match-numbers 2)
18 (match-numbers 4)
19
20
21 (define (match-type y)
22   (match y
23     ((? string? y) "I'm a string!")
24     ((? number? y) "I'm a number!")
25     ((? boolean? y) "I'm a boolean!")
26     ((? list? y) "I'm a list!")
27     (_ "I'm something else!")))
28
29 (match-type 5)
30 (match-type "hi")
31 (match-type #\c)
32
33 (define (get-last lst)
34   (match lst
35     ((list a) a)
36     ((list a b) b)
37     ((list a b c) c)))
38
39 (get-last (list 4 5 6))
40 (get-last (list 2))
41
42 (define (get-last-2 lst)
43   (match lst
44     ((list a) a)
45     ((list a b) b)
46     ((list _ _ c) c)))
47
48 (get-last-2 (list 1 2 3))
49
50 (define tenlist (list 1 2 3 4 5 6 7 8 9 10))
51
```

```

52 (match tenlist
53   ((list 1) "length 1")
54   ((list n ... 10) "last item is 10"))
55
56 (define (lotsa-strings str)
57   (match str
58     ((list _ (? string? x) ... y) (append x (list y)))
59     (_ void)))
60
61 (lotsa-strings (list "cats" "love" "pickles"))
62 (lotsa-strings (list "pickles"))
63 (lotsa-strings (list "cats" "love" 5 "pickles"))
64
65 (define (get-last-3 lst)
66   (match lst
67     ((list _ ... a) a)))
68
69 (get-last-3 (list 1))
70 (get-last-3 (list 1 2 3 4))
71 (get-last-3 (list 1 2))
72 ;(get-last-3 (list ))
73
74 (define (palindrome-checker lst)
75   (match lst
76     ((list a) #t)
77     ((list a a) #t)
78     ((list a b a) #t)
79     ((list a b b a) #t)
80     ((list a b c b a) #t)
81     (else #f)))
82
83 (palindrome-checker (list 1))
84 (palindrome-checker (list 1 2))
85 (palindrome-checker (list 1 2 3 2 1))
86
87 (define (dups lst)
88   (match lst
89     ((list (? string? x) _ ... x _ ...) #t)
90     (else #f)))
91
92 (dups (list "cat" "eats" "pickles"))
93 (dups (list "cat" "eats" "cat" "pickles"))
94 (dups (list "cat" "cat"))
95
96 (define (string-multiply str rep)
97   (letrec ((helper (lambda (s n res)
98                     (if (= n 0)
99                         res
100                        (helper s (- n 1) (string-append res s))))))
101     (helper str rep "")))
102

```

```
103 (string-multiply "cat" 5)
104
105 (expt 5 2)
106
107 (define (generic-power x n)
108   (match x
109     ((? string? y)(string-multiply y n))
110     ((? number? y)(expt y n))
111     (_ (printf "ERROR: WRONG DATA"))))
112
113 (generic-power 5 2)
114 (generic-power "cat" 2)
115 (generic-power #t 2)
```