1) Relacione as colunas. (_____ number? [1 2 3 \a :z 10 20 \b]) => _____

col 1	col 2
filter	(\a :z 10 20 \b)
remove	true
take-while	(1 2 3 10 20)
drop-while	(\a :z \b)
some	(1 2 3)

2) Relacione as colunas. (filter _____ '(1 \a -5 12 :b [:c :d] 0 -3)) => _____

col 1	col 2
number?	(:b)
char?	([:c :d])
keyword?	(1 -5 12 0 -3)
coll?	(\a)

3) Relacione as colunas. (filter _____ '(1 -2 7 10 -5 0 29 41)) => _____

col 1	col 2
zero?	(1 7 -5 29 41)
odd?	(1 7 10 29 41)
even?	(-2 -5)
pos?	(0)
neg?	(-2 10 0)

4) Relacione as colunas. (remove _____ '([1 2] \b {:m 0} () 7 d :z)) => _____

col 1	col 2
#(and (number? %) (odd? %))	([1 2] \b {:m 0} 7 :z)
#(or (symbol? %) (list? %))	({:m 0} () 7 d :z)
#(and (map? %) (contains? % :m))	([1 2] \b {:m 0} () d :z)
#(or (vector? %) (char? %))	([1 2] \b () 7 d :z)

5) Relacione as colunas. col1 => col2

(conj [\x \y] \a)	(\a \x \y)
(conj '(\x \y) \a)	(\a \x \y)
(cons \a [\x \y])	[\x \y \a]

6) Relacione as colunas.

(into _____ '(0 2 4 6)) => res

col 1	^-- col 2 são os elementos de res na ordem
[1 2]	0 1 4 6 2
'(1 2)	1 2 0 2 4 6
#{1 2}	6 4 2 0 1 2

7) Relacione as colunas. (_____ vector [0 1 2]) => _____

col 1	col 2
map	([0 0] [1 1] [2 2])
mapv	([0] [1] [2])
mapcat	[[0] [1] [2]]
map-indexed	(0 1 2)

8) Relacione as colunas. (take 5 _____) => _____

col 1	col 2
(repeatedly rand)	(0.87 0.93 0.42 0.40 0.53)
(repeat (rand))	(10 3.98 1.62 0.52 0.51)
(iterate rand 10)	(0.75 0.75 0.75 0.75 0.75)

9) Qual a diferença?

```
(for [x [1 3 5 7 2 4 6 15 17 19]] x)
(for [x [1 3 5 7 2 4 6 15 17 19] :when (odd? x)] x)
(for [x [1 3 5 7 2 4 6 15 17 19] :while (odd? x)] x)
```

10) (reduce #(conj % (f %2)) [] c) corresponde a (map f c) ?

11) Qual o resultado?

```
(reduce (fn [acc v] (if (> v acc) acc (+ 3 v))) [10 9 8 5 4 5])
```

Explique.

12) Relacione as colunas. col1 => col2

```
(map #(vector % %2) [1 2 3] [\a \b])                        ([1 \b] [1 \a])
(for [x [1 2 3] y [\a \b] :when (even? x)] (vector x y))    ([1 \a] [2 \b])
(reduce #(conj % (vector 1 %2)) '() [\a \b])                ([2 \a] [2 \b])
```

13) Defina filter com base em reduce.

14) apply : qual a diferença?

```
(max 4 7 2 8 0 5 1)
(apply max [4 7 2 8 0 5 1])
```

15) Relacione as colunas. (_____ {:a 1 :b 2} ____ ____) => {:a 2 :b 2}

col 1	col 2
assoc	:a inc
update	:a 2
update-in	[:a] inc

16) Dado
{:x {:m 3, :n 6}, :y 4, :z {:i {:f -1, :g 0}, :k 2}}
Como gerar
{:x {:m 3, :n 6}, :y 4, :z {:i {:f -1, :g 1}, :k 2}}
usando uma única chamada de update-in?

17) Dado um vetor de maps de coordenadas, filtrar aqueles que colidem. Há colisão quando $|x_2 - x_1| < 30$ e $|y_2 - y_1| < 30$.

Exemplo
```
(remover-colisoes [{:x 20 :y 80} {:x 75 :y 70} {:x 60 :y 50}])
;=> [{:x 20 :y 80}]
```

Projeto do fluxo de dados
- Criar um predicado colidem que compara dois maps num vetor.
- (colide [{:x 100 :y 200} {:x 50 :y 200}])
- ;=> false
- (colide [{:x 10 :y 10} {:x 20 :y 15}])
- ;=> true
- Fazer o produto cartesiano dos maps, sem (a,a).
- Obter colisões
- Coletar maps das colisões.
- Removê-las do vetor de maps original.

Sessão de REPL
É com você!