Racket cheatsheet.

`(if cond val-then val-else)` – în funcție de condiție, întoarce una dintre cele două valori..

- *Heads-up*: funcție nestrictă (macro) – nu evaluează decât condiția și valoarea care va fi întoarsă.

`(cond (cond1 val1) (cond2 val2) ...)` – if - else - if.

- întoarce $\texttt{val}_i$ pentru cel mai mic i pentru care $\texttt{cond}_i$ este adevărată.
- *Folosesc când*: am mai multe condiții cu valori întoarse independente unele de altele.
- *Heads-up*: funcție nestrictă (macro) – nu evaluează decât condițiile până la i inclusib și valoarea care va fi întoarsă.
- *Hint*: pentru ultima condiție (valoarea default) se poate folosi `#t` sau `else`.
- *Heads-up*: dacă $\nexists i$ a.î. $\texttt{cond}_i$ nu este `#f`, se întoarce `#<void>`.

`(reverse L)` – inversează o listă.

`(append L1 L2)` – concatenează două liste.

`(list e1 e2 ...)` – creează o listă care va conține elementele date.

- *Heads-up*: `(list x)` nu convertește pe `x` la o listă, ci construiește o listă care va conține pe `x` ca singur element.

`(member e L)` – Caută un element într-o listă.

- dacă $e \notin L$, întoarce `#f`
- dacă $e \in L$, întoarce sublista din L care începe cu cea mai din stânga apariție a lui e.

`(assoc e LP)` – Caută într-o listă de perechi după primul element al perechii.

- *Heads-up*: funcționează doar dacă lista este o listă de perechi (și listele sunt perechi). Aruncă excepție dacă înainte de a se întoarce găsește în listă ceva care nu este o pereche.
- dacă $e \notin$ `(map car L)`, întoarce `#f`
- dacă $e \in$ `(map car L)`, întoarce prima pereche din L care are pe e ca prim element.

`(andmap f L)` – verifică dacă toate aplicările lui f pe elemente din L dau diferit de `#f`.

- este *teoretic* echivalent cu `(apply and (map f L))` (dar `and` nu poate fi dat lui `apply` pentru că este macro.
- *Heads-up*: funcție nestrictă (macro) – evaluează f pe elemente din e doar atât timp cât f nu întoarce `#f`.
- *Heads-up*: dacă nu întoarce `#f`, atunci întoarce `(f` $\texttt{e}_{n-1}$`)` cu n lungimea listei.

`(ormap f L)` – verifică dacă vreuna dintre aplicările lui f pe elemente din L dă diferit de `#f`.

- este *teoretic* echivalent cu `(apply or (map f L))` (dar `or` nu poate fi dat lui `apply` pentru că este macro.
- *Heads-up*: funcție nestrictă (macro) – evaluează f pe elemente din e doar atât timp cât f întoarce `#f`.
- *Heads-up*: întoarce `(f` $\texttt{e}_i$`)` pentru cel mai mic i pentru care `(f` $\texttt{e}_i$`)` întoarce diferit de `#f`.

`(map f L)` – aplică o funcție fiecăruia dintre elementele unei liste.

- dacă `L` ← `(list e1 e2 ...en)`, atunci `(map f L)` → `(list (f e1) (f e2) ...(f en))`
- *Matematic*: $map(f, L) = [f(e) \mid e \in L]$
- *Folosesc când*: doresc să obțin o listă L' în care fiecare element este prelucrarea prin f a unui element din L. Prelucrarea fiecărui element se va face independent de prelucrarea celorlalte elemente.
- *Heads-up*: rezultatul este întotdeauna o listă
- *Heads-up*: rezultatul este întotdeauna de aceeași lungime cu lista dată

`(map f L1 L2 ...)` – aplică o funcție peste elementele corespondente din listele date ca argument.

- dacă `L1`← `(list e11 e12 ... e1n)`, ..., `Lm`← `(list em1 ... emn)`,
  atunci `(map f L1 L2 L3)` → `(list (f e11 ... em1) ... (f e1n ... emn))`
- *Matematic*: $map(f, L_1, L_2, \ldots L_m) = [f(e_{1i}, e_{2i}, \ldots\ e_{mi}) \mid i = \overline{1,n}, e_{ji} \in L_j]$
- *Heads-up*: listele trebuie să fie de aceeași lungime
- *Heads-up*: rezultatul este întotdeauna o listă
- *Heads-up*: rezultatul este întotdeauna de aceeași lungime cu listele date

`(filter f L)` – selectează elementele pentru care o funcție întoarce adevărat.

- *Tehnic*: sunt selectate elementele pentru care `f` nu întoarce `#f`
- *Matematic*: $filter(f, L) = [e \mid e \in L \wedge f(e) \neq$ `#f`$]$
- *Folosesc când*: doresc să obțin o listă L' care conține o parte din elementele listei L, și anume acele elemente pentru care este îndeplinită o condiție.
- *Heads-up*: rezultatul este întotdeauna o listă
- *Heads-up*: în rezultat, elementele nu sunt prelucrate față de lista L (dacă un element apare în rezultat, apare la fel ca și în lista L, indiferent ce face funcția f).

`(foldl f v0 L)`, `(foldr f v0 L)` – realizează o prelucrare mai complexă peste elementele unei liste.

- dacă `L` ← `(list e1 e2 ...en)`, atunci `(foldl f v0 L)` → `(f en (...(f e2 (f e1 v0))))`
- iar `(foldr f v0 L)` → `(f e1 (f e2 (...(f en-1 (f en v0)))))`
- *Folosesc când*: am o listă în care elementele trebuie prelucrate în ordine, dar am nevoie de o prelucrare pe care nu o pot face cu `map` sau `filter`.
- *Heads-up*: rezultatul poate avea orice tip
- *Heads-up*: dacă fold este aplicat pe o singură listă, atunci funcția f ia 2 argumente
- *Cum implementez folosind foldl*:

  `(foldl (λ (ei r)` ← $e_i \in L$, r – rezultatul prelucrării elementelor $e_1 \ldots e_{i-1}$
  `produc rezultatul pentru elementele e1...ei`
  `)`
  `v0 L)` – v_0 este valoarea pentru când L este vidă, L este lista.

- *Cum implementez folosind foldr*:

  `(foldr (λ (ei r)` ← $e_i \in L$, r – rezultatul prelucrării elementelor e_n, $e_{n-1} \ldots e_{i+1}$
  `produc rezultatul pentru elementele en...ei`
  `)`
  `v0 L)` – v_0 este valoarea pentru când L este vidă, L este lista.

`(foldl f` v_0 $L_1 \ldots L_m$`), (foldr f` v_0 $L_1 \ldots L_m$`)` – Fold pe mai multe liste.

- la fel ca și cu o singură listă, dar funcția dată lui fold primește n+1 argumente (ultimul fiind acumulatorul / rezultatul parțial), unde lungimea listelor date este n.
- *Heads-up*: listele trebuie să fie de aceeași lungime.