Structuur van Computerprogramma's I

Examen Eerste Zit

Academiejaar 2025-2026

Maandag 19 januari 2026

Instructies en Regels:

- Dit examen is **gesloten boek**.
- Neem voor elke vraag een **volledig nieuw blad** en schrijf op elk blad je naam, studierichting en het nummer van de vraag.
- **Antwoorden in potlood worden niet aanvaard!** Lees elke vraag volledig en aandachtig voor je aan je oplossing begint!
- Je hebt **3 uur** de tijd.
- Maak in je oplossingen **geen** gebruik van destructieve operaties (zoals `set!`, `set-car!`, etc.) tenzij de vraag dit expliciet toestaat of vereist.

Veel succes!

1 Lijsten

Context: Jean-Pierre werkt in de bibliotheek en heeft twee stapels steekkaarten die elk al alfabetisch gesorteerd zijn op auteursnaam (gerepresenteerd door getallen voor de eenvoud). Hij wil deze twee stapels samenvoegen tot één grote, gesorteerde stapel zonder alles opnieuw te moeten sorteren.

a) Schrijf een procedure `(merge stapel1 stapel2)` die twee **gesorteerde** lijsten van getallen samenvoegt tot één nieuwe gesorteerde lijst.

- Je mag **geen** gebruik maken van de ingebouwde procedure `append`.
- Je mag ervan uitgaan dat de inputlijsten correct gesorteerd zijn (klein naar groot).

```
1 > (merge '(1 3 5) '(2 4 6))
2 (1 2 3 4 5 6)
3 > (merge '(1 4 6 9) '(2 4))
4 (1 2 4 6 9)
```

b) Levert je oplossing voor a) een recursief of een iteratief proces op? Leg in je eigen woorden uit waarom je dit antwoord geeft (argumenteer bondig!).

c) Schrijf nu de **andere versie** voor a) die dezelfde lijst teruggeeft.

- Als je in a) een recursief proces schreef, schrijf nu de versie die een iteratief proces genereert.
- Als je in a) een iteratief proces schreef, schrijf nu de versie die een recursief proces genereert.

2 Hogere-orde Procedures

Context: Voor een wetenschappelijk experiment willen we functies kunnen uitvoeren die "veilig"zijn en statistieken bijhouden. Als een functie faalt (omdat de input ongeldig is), mag het programma niet crashen, maar moet dit geregistreerd worden.

a) Schrijf de procedure `(encapsulate proc pred?)`. Deze procedure neemt twee argumenten: een functie `proc` en een voorwaarde `pred?`. De procedure moet een nieuwe procedure retourneren die als volgt werkt:

- Als de input voldoet aan `pred?`, wordt `proc` uitgevoerd.
- Als de input **niet** voldoet, wordt het symbool `'error` teruggegeven.
- De procedure houdt intern bij hoeveel keer ze succesvol was en hoeveel keer ze faalde.
- Via de berichten `'success` en `'failure` kunnen deze tellers opgevraagd worden.

```
1 > (define safe-sqrt (encapsulate sqrt positive?))
2 > (safe-sqrt 25)  ; -> 5
3 > (safe-sqrt -5)  ; -> 'error
4 > (safe-sqrt 'success) ; -> 1
5 > (safe-sqrt 'failures) ; -> 1
6 > (safe-sqrt 25)  ; -> 5
7 > (safe-sqrt 'success) ; -> 2
```

b) Ga ervan uit dat de standaard, recursieve procedure `fac` (faculteit) reeds globaal gedefinieerd is. Schrijf de Scheme-instructie(s) om de bestaande globale variabele `fac` te overschrijven met het resultaat van jouw `encapsulate`, zodat we de recursieve stappen kunnen tellen.

Let op: Je mag ***`define`*** *niet gebruiken om een variabele te overschrijven die al bestaat.*

3 Boomstructuren

Context: Bol.com heeft zijn categorieën herzien. De producten zijn georganiseerd in een boomstructuur. Elke knoop is een lijst waarbij het eerste element de naam van de categorie is, en de rest van de lijst de subcategorieën zijn.

```
(define catalogus
  '(webshop
     (elektronica
       (gaming (consoles) (accessoires))
       (audio (koptelefoons) (speakers)))
     (kledij
       (heren (broeken) (shirts))
       (dames (jurken) (rokken)))
     (boeken (fictie) (non-fictie))))
```

a) Schrijf het predicaat `(sub-categorie? boom cat1 cat2)`. Deze procedure moet nagaan of `cat2` een (directe of indirecte) subcategorie is van `cat1` binnen de gegeven `boom`.

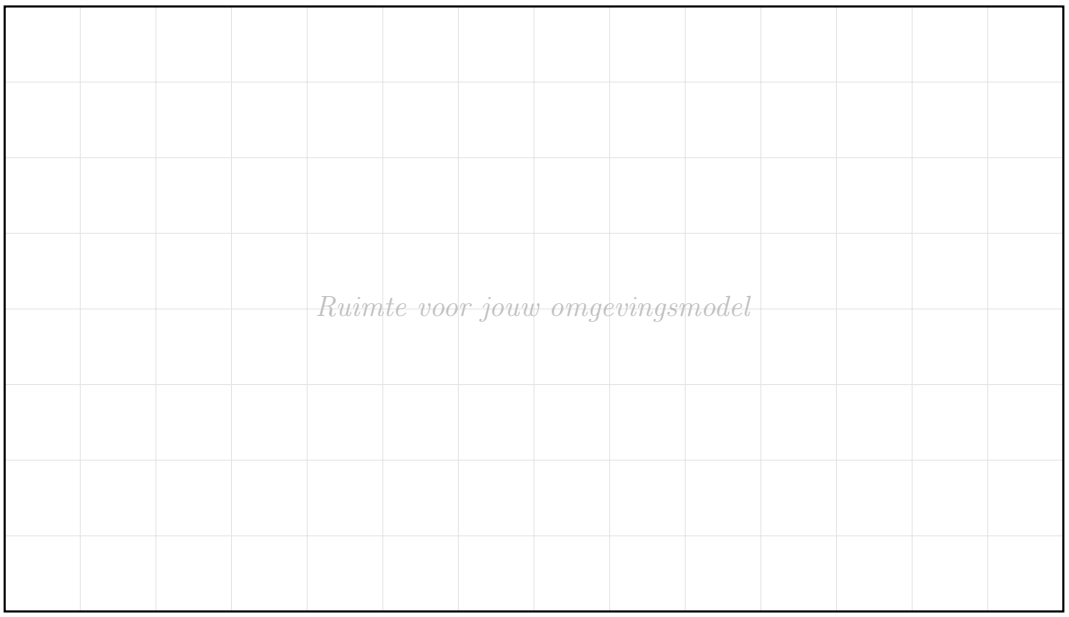
```
> (sub-categorie? catalogus 'kledij 'broeken)    ; -> #t
> (sub-categorie? catalogus 'webshop 'consoles)  ; -> #t
> (sub-categorie? catalogus 'broeken 'shirts)    ; -> #f
```

4 Omgevingsmodellen

Bestudeer de volgende codeaansnede aandachtig:

```
1  (define (second a b next)
2    (let ((a (next b)))
3      (+ a b)))
4
5  (define (first input)
6    (let* ((i 1) (j 10))
7      (second input i (lambda (x)
8                        (set! i (+ i 1))
9                        (+ x j)))))
10
11 ;; De aanroep:
12 (first 0)
```

Teken het volledige omgevingsmodel (environment model) dat ontstaat bij de evaluatie van de aanroep `(first 0)`. Geef duidelijk aan waar de omgevingswijzers (environment pointers) naar wijzen en wat de return values zijn. Wat is de uiteindelijke output?



Output: _____

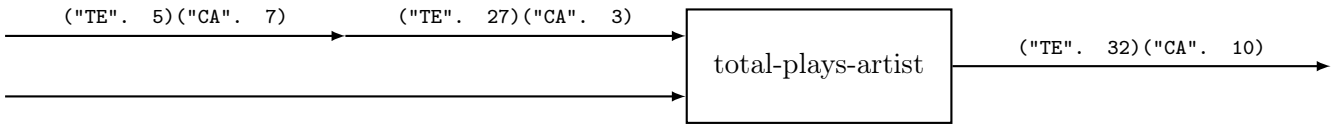
5 Stromen (Streams)

Context: Als data-engineer bij Spotify krijg je een constante stroom van luistergegevens binnen. Deze data is georganiseerd als een **"stroom van stromen"**. Elke interne stroom bevat het luistergedrag van één gebruiker als paren van `(Artiest . AantalPlays)`.

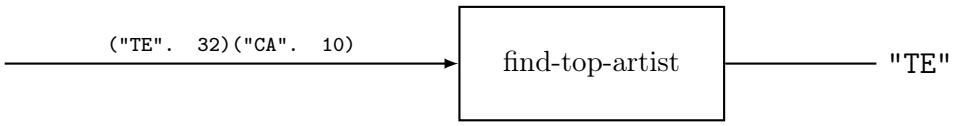
Je mag gebruik maken van de standaard stroom-operatoren: `the-empty-stream`, `empty-stream?`, `cons-stream`, `head`, `tail`, `take`, `stream-filter`, `stream-map`, `stream-accumulate`, `stream-accumulate-n`, `stream-for-each`, `stream-append`, `stream-flatten`, `stream-interleave`, `stream-print`, `stream-accumulate` `stream-interleave-delayed`, `stream-flatten/interleaved`, `stream-values`, `enumerate-tree`, `enumerate-int`, `pairs`.

a) Schrijf de procedure `(total-plays-artist users-stream)`. Deze procedure neemt de stroom van gebruikers (een stroom van stromen) en moet één enkele stroom teruggeven die de totalen per artiest bevat (gesommeerd over alle gebruikers).

Aanname: De volgorde van artiesten is in elke stroom identiek, en al de artisten zitten in alle stromen.



b) Schrijf een procedure `(find-top-artist total-stream)` die de artiest met de meeste plays uit de berekende totaal-stroom haalt.



6 Theorie

Beantwoord de volgende meerkeuzevragen. Kies het juiste alternatief (A, B, C of D). Indien het antwoord "Dïs, vul dan het juiste begrip of antwoord zelf in. *(Dit onderdeel telt voor 5 van de 20 punten en bestaat in totaal uit 10 vragen, waarvan hieronder de eerste 4 zijn opgenomen.)*

1. **Scoping Regels**
   Scheme maakt standaard gebruik van Lexical Scoping. Wat betekent dit concreet voor de waarde van vrije variabelen in een procedure?

   A. De waarde wordt gezocht in de omgeving waarin de procedure wordt *aangeroepen* (runtime).

   B. De waarde wordt bepaald door de omgeving waarin de procedure werd *gedefinieerd* (definition time).

   C. De waarde is altijd globaal en kan niet overschreven worden door lokale bindings.

   D. *Geen van bovenstaande, namelijk:* _____

2. **Recursie en Processen**
   Wanneer genereert een recursief gedefinieerde procedure (syntactische recursie) een iteratief proces in het geheugen?

   A. Wanneer de procedure gebruik maakt van `set!` om een teller bij te houden.

   B. Wanneer de recursieve oproep de allerlaatste actie is in de body van de procedure (tail recursion).

   C. Wanneer de procedure wordt uitgevoerd binnen een `let`-omgeving.

   D. *Geen van bovenstaande, namelijk:* _____

3. **Unit Testing**
   Wat is het primaire doel van Unit Testing in softwareontwikkeling?

   A. Het testen van de samenwerking tussen alle modules van het systeem (integratie).

   B. Het verifiëren van de correctheid van de kleinste testbare onderdelen (units) in isolatie.

   C. Het meten van de snelheid en performantie van het volledige algoritme.

   D. *Geen van bovenstaande, namelijk:* _____

4. **Memoization**
   Wat is het belangrijkste voordeel van het toepassen van Memoization op een procedure zoals de Fibonacci-reeks?

   A. Het vermindert het geheugengebruik drastisch door variabelen te verwijderen.

   B. Het voorkomt exponentiële rekentijd door resultaten van eerdere oproepen op te slaan en te hergebruiken.

   C. Het maakt de code destructief waardoor deze makkelijker te debuggen is.

   D. *Geen van bovenstaande, namelijk:* _____

... (10 Totaal) ...