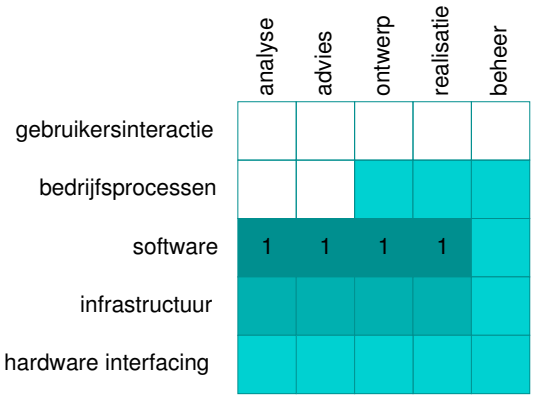
HOGESCHOOL ROTTERDAM / CMI

Functioneel programmeren 1

TINFUN01

Aantal studieunten:	3 ects
Modulebeheerder:	Giuseppe Maggiore

Modulebeschrijving

Modulenaam:	Functioneel programmeren 1
Modulecode:	TINFUN01
Aantal studiepunten en studiebelastinguren:	Deze module levert 3 ectsstudiepunten op, hetgeen overeenkomt met 84 uur. • 8 × 120 minuten hoorcollege • 8 × 120 minuten practicum • 12 × 120 minuten zelfstudie
Toetsing:	Practicumopdrachten
Werkvorm:	Hoorcollege en practicum
Vereiste voorkennis:	Alle modules wiskunde en programmeren uit het eerste jaar en de eerste helft van het tweede jaar.
Leermiddelen:	• Boek: Types and Programming Languages, auteur: Benjamin Pierce • Boek: Friendly F# (Fun with game programming Book 1), auteurs: Giuseppe Maggiore, Giulia Costantini • Text editors: Emacs, Notepad++, Visual Studio, Xamarin Studio, etc.
Draagt bij aan competenties:	
Leerdoelen:	• Het kunnen adviseren over en analyseren, ontwerpen en realiseren van een computerprogramma op niveau 1 (leerdoel A). • Het kunnen ontwerpen, schrijven, compileren en uitvoeren van een correct werkend ML programma (leerdoel O). • Het in correct Nederlands en met gebruikmaking van het juiste jargon kunnen communiceren over de programmeertaal (leerdoel C).

Inhoud:	• Programmeerparadigma's; • Waarden en `let`; • Functies en recursieve functies • Data structuren; • Recursieve data structuren: lijsten, bomen; • Hogere orde functies: `map`, `filter`; • Abstractie door records van functies; • Staart recursie, accumulatoren, *continuation passing style*.
Modulebeheerder:	Giuseppe Maggiore
Datum:	13 april 2015

1 Algemene omschrijving

1.1 Inleiding

Functioneel programmeren is een in opkomst zijnde manier van programmeren die nogal wat verschillen vertoont met de klassiek imperatieve manier van programmeren zoals dit bekend is in programmeertalen als C, C++, Java e.v.a.

Functionele talen zijn gebaseerd op de zogeheten λ *calculus*, waardoor deze talen fundamenteel anders werken dan de meeste programmeurs gewend zijn. Daar puur functionele talen geen side effects kennen en bovendien referential transparency gebruiken i.p.v. destructive update betekent dat programma's geschreven met functionele talen hebben hogere textitencapsulation van functies en data structuren, en vervolgens minder bugs en algemene fouten. Met het steeds populairder worden van deze platformen wordt ook het functioneel programmeren populairder.

Verder is te zien dat met het steeds verder toenemen van het abstractieniveau, waarop computerprogramma' s functioneren de behoefte aan talen, waarin deze abstracties compact kunnen worden uitgedrukt, toeneemt. Functionele talen lijken hier tot nu toe de betere papieren voor te hebben.

1.2 Relatie met andere onderwijseenheden

Op deze module wordt voortgebouwd in de module TINFUN02.

1.3 Leermiddelen

Verplicht:

- Presentaties die gebruikt worden in de hoorcolleges (pdf): te vinden op N@tschool
- Opdrachten, waaraan gewerkt wordt tijdens het practicum (pdf): te vinden op N@tschool

Facultatief:

- Boek: Types and Programming Languages, auteur: Benjamin Pierce
- Boek: Friendly F# (Fun with game programming Book 1), auteurs: Giuseppe Maggiore, Giulia Costantini
- Text editors: Emacs, Notepad++, Visual Studio, Xamarin Studio, etc.

2 Programma

De volgende lijst zijn de onderwerpen van de cursus:

1. Inleiding, interpreter, eerste programma;
2. Let, let-rec, basale flow-controle;
3. Primitive types, tuples, type sommen, pattern-matching;
4. Recursieve data-structuren, lijsten;
5. Records;
6. Hogere-orde-functies (HOF's);
7. Functie samenstelling: records-van-lambda's, continuation-passing-style (CPS);

3 Toetsing en beoordeling

3.1 Procedure

Deze module wordt getoetst middels een aantal practicumopdrachten. Deze opdrachten zijn te vinden op N@tschool.
Voorwaarden en uitgangspunten:

- De practicumopdrachten bepalen het eindcijfer.
- De practicumopdrachten bestaan uit een game/simulatie voorbeeld die is niet compleet of foutieve; studenten moeten elke voorbeeld uitbreiden of corrigeren totdat het werkt volgens de opdracht beschrijving.
- De practicumopdrachten moeten documentatie bevatten.

Voor deze manier van toetsen is gekozen om de volgende redenen:

- Door de gegeven bronnen studenten moeten code lezen en begrijpen (leerdoel A).
- Door deze bronnen te corrigeren of invullen moeten studenten code schrijven (leerdoel O).
- Door documentatie schrijven moeten studenten communiceren over hun code (leerdoel C).

De cijfer van elke practicum opdracht is bepaald door:

- Correctheid van puur, functionele code (gebruik van `mutable` and `ref` is verboden) (60%).
- Compleetheid en helderheid van documentatie (20%).
- Gebruik van functionele patronen en idiomen zoals gezien in de lessen (20%).

3.2 Opdrachten

3.2.0.1 Opdracht 1 - let, expressies, definitie van functies Studenten moeten twee functies definiëren om de positie en snelheid van een object te bewerken. De eindresultaat is dat het object door het scherm valt met zwaartekrachtversnelling.
De signatures van de functies en waarden dat moeten geïmplementeerd worden zijn:

```
updatePosition (y:float) (vy:float) (dt:float) : float
updateVelocity (y:float) (vy:float) (dt:float) : float
```

3.2.0.2 Opdracht 2 - tupels, if's Studenten moeten één enkele functie definiëren om de positie en snelheid van een object te bewerken. De functie retourneert een tupels met twee elementen: de nieuwe positie en de nieuwe snelheid van het object. Als het object de rand van het scherm raakt, dan moet hij opspringen.
De signatures van de functies en waarden dat moeten geïmplementeerd worden zijn:

```
updateBall (y:float, vy:float) (dt:float) : float * float
```

3.2.0.3 Opdracht 3 - records Studenten moeten twee records definiëren: `Vector2` en `MovingObject`. De `MovingObject` record bevat de positie en de snelheid van een object. Positie en snelheid zijn instanties van `Vector2`. Het object moet bewerkt worden zoals in *Opdracht 2*, maar nu met 2D beweging.
De signatures van de functies en waarden dat moeten geïmplementeerd worden zijn:

```
initialBall : Ball
updateBall (b:Ball) (dt:float) : Ball
```

3.2.0.4 Opdracht 4 - lijsten, pattern matching

Studenten moeten een recursieve functie definiëren om een lijst van `MovingObject`'s te bewerken. De functies van *Opdracht 3* mogen gebruikt worden. Objecten die vallen buiten de randen van het scherm moeten van de lijst verwijdert worden.
De signatures van de functies en waarden dat moeten geïmplementeerd worden zijn:

```
initialBalls : List<Ball>
rec updateBalls (balls:List<Ball>) (dt:float) : List<Ball>
rec removeOutOfBounds (balls:List<Ball>) : List<Ball>
```

3.2.0.5 Opdracht 5 - lijsten, hogere orde functies

Studenten moeten de functie van *Opdracht 4* weer schrijven, maar door:

- eigen `map` en `filter` functies
- de standaard `map` en `filter` functies

De signatures van de functies en waarden dat moeten geïmplementeerd worden zijn:

```
rec map' (f:'a->'b) (l:List<'a>) : List<'b>
rec filter' (f:'a->bool) (l:List<'a>) : List<'a>
rec updateBalls' (balls:List<Ball>) (dt:float) : List<Ball>
rec removeOutOfBounds' (balls:List<Ball>) : List<Ball>

rec updateBalls''' (m:(Ball -> Ball) -> List<Ball> -> List<Ball>) (balls:List<Ball>) (dt
    :float) : List<Ball>
rec removeOutOfBounds''' (f:(Ball -> bool) -> List<Ball> -> List<Ball>) (balls:List<Ball
    >) : List<Ball>
```

3.2.0.6 Opdracht 6 - som types

Studenten moeten een 2D zoekboom (*quadtree*) definiëren. Door dit zoekboom wordt het bepalen van collisies tussen kogels en asteroïden veel sneller ($O(N \log_4 N)$ in plaats van $O(N^2)$).
De signatures van de functies en waarden dat moeten geïmplementeerd worden zijn:

```
type AsteroidQuadTree
rec treeToSeq (t:AsteroidQuadTree) : List<AsteroidQuadTree>
rec insert (a:Asteroid) (q:AsteroidQuadTree) : AsteroidQuadTree
rec existsColliding (p:Projectile) (q:AsteroidQuadTree) : bool
```

3.2.0.7 Opdracht 7 - abstracte datatypes

Studenten moeten een generieke collisie detectie functie. De functie bepaalt collisies tussen verschillende soorten van objecten. Om de vorm en positie van de objecten te bepalen gebruiken we een record van functies die specificeert hoe de generieke objecten uitgepakt moeten worden.
De signatures van de functies en waarden dat moeten geïmplementeerd worden zijn:

```
let genericUpdate<'ship, 'asteroid, 'projectile>
  (s_ops:EntityOperations<'ship>)
  (a_ops:EntityOperations<'asteroid>)
  (p_ops:EntityOperations<'projectile>)
  (ship : 'ship, asteroids : List<'asteroid>, projectiles : List<'projectile>) (dt:float
      )
  : 'ship * List<'asteroid> * List<'projectile>

let genericDraw<'ship, 'asteroid, 'projectile>
  (s_ops:EntityOperations<'ship>)
  (a_ops:EntityOperations<'asteroid>)
  (p_ops:EntityOperations<'projectile>)
  (spriteBatch:SpriteBatch)
  (ship : 'ship, asteroids : List<'asteroid>, projectiles : List<'projectile>)
  : Unit


type Ship
shipOperations : EntityOperations<Ship>

type Asteroid
```

```
asteroidOperations : EntityOperations<Asteroid>

type Projectile
projectileOperations : EntityOperations<Projectile>
```

Note: the above signatures are just a starting point. The signatures, and the rest of the downloadable sources, may need to be adjusted in order to be made to work.

3.2.0.8 Opdracht 8 - continuation passing style (CPS) Studenten moeten twee kleine CPS functies schrijven: `faculteit` en `fibonacci`. Studenten moeten ook de zoekprocedure van *Opdracht 6* weer schrijven met een variant van CPS die maakt het mogelijk om de procedure uit te voeren tot een aantal stappen (*delayed/concurrent execution*).
Note: no signatures or code is provided. As a very advanced assignment you are required to set up your own project from scratch.

3.3 Cijfers

De eerste vijf opdrachten zijn verplicht. Met deze opdrachten het is mogelijk om een zes cijfer te halen. Opdrachten zes en zeven zijn uitsluitend geldig als de eerste vijf opdrachten zijn correct. Met opdrachten zes en zeven de maximum cijfer mogelijk wordt een negen. De laatste opdracht is zeer moeilijk, en maakt het mogelijk om een tien te halen.

Bespreken van toetsopdrachten

Herkansing (ontbrekende inleveren in week N van volgende periode)

Bijlage 1: Toetsmatrijs

	~~Leerdoelen~~ Inhoud	Dublin descriptoren
1	editor, compiler, source, binary, statements, declaratieve/imperatieve talen	1,2,3,4
2	waarden: integer, float, double, boolean, char, String	1,2,3,4
3	selectie: if, if-else, if-else-if, match-with	1,2,3,4
4	recursie	1,2,3,4
5	lijsten en recursieve data structuren	1,2,3,4
6	HOF's en hun handeling	1,2
7	lambda calculus	1

Dublin-descriptoren:

1. Kennis en inzicht
2. Toepassen kennis en inzicht
3. Oordeelsvorming
4. Communicatie