1 Kenmerken

1.1 Haskell

- Puur functioneel
- Sterk en statisch getypeerd

1.2 Bash

- Imperatieve programmeertaal
- Handig om snel shell scriptjes mee te schrijven
- Boven ieder bestand moet de shebang staan

1.3 Prolog

- Decleratieve programmeertaal (Specificeer wat het probleem is in plaats van het op te lossen)
- Krachtig voor KI, databases, taalverwerking en puzzels
- Bestaat uit Facts (dingen die als waar worden gedefinieerd) en Rules (dingen die mogelijk zijn op basis van feiten of andere regels)

1.4 Python

- Abstracte taal weg van machine details
- Meer directe expressies van mentale primitieven
- Geforceerde restrictieve stijl (indentaties etc)
- Boven ieder bestand moet een shebang staan

1.5 Erlang

- Gedistribueerd en functioneel (functionele taal)
- Dynamisch en sterk getypeerd
- Fout tolerant (let is crash!)
- Prolog-achtige syntax

1.6 Go

- Gedistribueerd en imperatief
- Statisch en sterk getypeerd

1.7 C++

- Object georiënteerd
- Op een generieke manier programmeerbaar

2 Functies, Statements, Operatoren en Variabelen

2.1 Haskell

Een voorbeeld van het toekennen van variabelen in de terminal:
Prelude> Let foo = 3
Prelude> foo + 3
Prelude> 6

Een voorbeeld van een eigen geschreven optel functie in het bestand Samenvatting.hs:
module Samenvatting where
add :: Integer->Integer->Integer
add a b = a + b

Er bestaan if-statements, maar het wordt aangeraden om guards ('|') te gebruiken omdat dit zorgt voor beter leesbare code.
Dit kan er als volgt uitzien:
discribeLetter :: Char -> String
discribeLetter c | c >= 'a' && c <= 'z' = "Lower Case"
————————————| c >= 'A' && c <= 'Z' = "Upper Case"
————————————| otherwise————= "No ASCII Letter"

Je kan foldr en foldl gebruiken om rechtsom of linksom over lijsten te itereren, dit ziet kan er in de terminal zo uit zien:

Prelude> foldr (+) 0 [1,2,3,4,5]
15

Wat hier gebeurd is het volgende, er wordt, vanaf rechts beginnend, steeds de waarde bij het begin getal (eerst 0 dan 1, dan 3, etc) opgeteld.

2.2 Bash

Functies worden als volgt aangemaakt:
function Samenvatting{
}
Functies worden als volgt aangeroepen in de code:
Samenvatting Argument1 Argument2 (dus net zo als in de terminal)

Na het gebruik van variabelen kan je deze weer leeg maken door gebruik te maken van: unset varname.

De if-statement ziet er als volgt uit:
if [[expressie = iets]]; then
—Doe iets
else
—Doe iets anders
fi

Dus: blokhaken ipv gewone haakjes en fi om de if-statement af te sluiten.

De for-loop ziet er als volgt uit:
for i in Variabele
do
—doe iets
done

Het schrijven van bestanden gaat als volgt:
echo "Overschrijven" > bestand.txt verwijderd alles en schrijft "Overschrijven" in het bestand
echo "Onderschrijven" » bestand.txt schrijft onder de laatste regel "Onderschrijven" in het bestand

Strings kunnen op de volgende manieren worden gemanipuleerd:

- $X = "aabbccaabbcc"
- echo length ${#X} = length 12 -> length
- echo ${X:0:1}, ${X:4}, ${X:-2} = a, ccaabbcc, cc -> cut
- echo ${X/a/Z}, ${X//a/Z} = Zabbccaabbcc, ZZbbccZZbbcc -> replace
- echo ${X#a*b}, ${X##a*b} = bccaabbcc, cc -> delete front
- echo ${X%b*c}, ${X%%b*c} = aabbccaab, aa -> delete back

2.3 Prolog

Atoms zijn (constante) opbjecten en beginnen met een kleine letter, bijvoorbeeld:

- olifant
- bob
- alice

Variabelen zijn objecten waarvan de naam met een hoofdletter of underscore begint, bijvoorbeeld:

- X
- Schedule
- _mother
- Anonieme variabele: _

Facts voorbeeld:
female(Alice).
male(Bob).
male(Kevin).

In terminal:
?- male(Bob).
true.

?- male(X).
X = Bob;
X = Kevin.

Rules voorbeeld:
parent(Alice, Bob).
parent(Bob, Kevin).

mother(X, Y) :- female(X), parent(X, Y). (-> X moet een vrouw zijn en ouder van Y als X de moeder van Y is)
father(X, Y) :- male(X), parent(X, Y). (-> X moet een man zijn en de ouder van Y als X de vader van Y is)

Let bij het controleren van gelijkheid goed op het volgende:

- = unificeert beide kanten: X = 2+2 -> X = 2+2
- == vergelijkt beide termen: 3+1 == 2+2 -> false
- =:= evalueert beide kanten: 3+1 =:= 2+2 -> true
- is evalueert de rechterkant en stelt de linkerkant gelijk aan die waarde: X is 2+2 -> X = 4

2.4 Python

Alle variabelen zijn directe referenties naar een waarde, zo geeft:
a = 15
b = a
a -> 15 & b -> 15

Bij lijsten gebeurt het volgende:
a = [15]
b = a
a[0] += 1
print (a, b)
print (id(a), id(b))
Dit geeft:
16 16
140053768683656 140053768683656

Dit kan echter voorkomen worden als er in plaats van b = a staat b = copy.deepcopy(a)

2.5 Erlang

Strings zijn lijsten van integers, dit ziet er in de terminal als volgt uit:
1> [97, 98, 99]
"abc"

Met _ wordt input in een functie genegeerd, bijvoorbeeld:
samenvatting(1) -> Het getal is 1;
samenvatting(_) -> Het getal is niet 1.

Ook kunnen er guards gebruikt worden:
old_enough(X) when X >= 18 -> true;
old_enough(_) -> false.

Case statements kunnen ook gebruikt worden:
lucky(X) ->
—case X of
— —4 -> lucky;
— —6 -> doomed;
— —7 -> very lucky;
— —_ -> not so lucky
—end.

Invoer kan gedaan worden met io:get_line() en uitvoer met io:format()

Er kunnen meerdere processen gecreëerd worden door processen te spawnen, deze processen kunnen informatie aan elkaar doorgeven en hiermee door werken. Het doorsturen van informatie gaat door middel van het sturen van berichten aan elkaar, er is echter geen garantie dat berichten ook aankomen in de volgorde waarop ze worden verstuurd.

2.6 Go

Variabelen worden buiten een functie als volgt gedeclareerd:
var i int 6 (eerst de naam, dan het type en dan de waarde)

Variabelen worden binnen een functie als volgt gedeclareerd:
i := 6

In een statement kunnen meerdere returnwaarden terug worden gegeven, bijvoorbeeld:
func verwissel(a, b int) (int, int){
—return b, a
}

func main(){
—fmt.Println(verwissel(4,3))
}

Dit geeft:
3, 4

De enige soort loop is de for-loop, hier kan echter wel alles mee worden gedaan.

Goroutines kunnen worden gebruikt om meerdere processen naast elkaar te laten lopen, hierdoor kunnen er dus meerdere functies tegelijkertijd worden uitgevoerd om snel oplossingen op problemen, zoals de weg uit een doolhof, te vinden. Goroutines kunnen gegevens met elkaar uitwisselen door het gebruik van channels.

Goroutines kunnen ook vastlopen, als alle goroutines vastlopen dan veroorzaakt dit een deadlock. Een manier om dit te voorkomen is door de routines alleen door te laten lopen als er bevestiging is gekomen dat de vorige uitkomst is aangekomen, als dit niet het geval is kan de routine sluiten en kan hij ook niet meer vastlopen.
Als een routine de taak heeft voltooid en meerdere uitkomsten zijn onmogelijk/niet nodig dan is het van belang dat alle channels worden gesloten doormiddel van: close(channelnaam)

2.7 C++

Door middel van de if, switch, condition operator is het makkelijk functies op een enkele regel te zetten, bijvoorbeeld:
int a = 1, b = 2
largest = a > b ? a : b (Is dit waar ? Ja : Nee)

Er zijn drie manieren om data door te geven aan functies:

- Een kopie mee geven (Geen speciale notatie en heeft geen effect buiten de functie)
- Een pointer mee geven (Een & bij het aanroepen en een * bij de functiebeschrijving, veranderd de waarde buiten de functie)
- Een reference mee geven (Een & bij de functiebeschrijving, veranderd de waarde buiten de functie)

Een pointer wijst direct op de waarde in het geheugen, terwijl een referentie dit niet doet. Hierdoor kan de data van een pointer gelijk zijn aan NULL en is dit bij een referentie niet mogelijk.

Operators in C++ kunnen overloaded worden, dit betekend dat het mogelijk is om operators, zoals de +, iets anders te laten doen dan waar ze eigenlijk voor bedoeld zijn.

Templates kunnen worden gebruikt om er voor te zorgen dat functies met dezelfde naam en functie niet voor alle soorten datatypen geschreven te hoeven te worden. Het is hiermee namelijk mogelijk om het datatype mee te geven, waardoor de functie altijd correct uitgevoerd kan worden.

3 Lijsten, Tuples en andere Datatypen

3.1 Haskell

Lijsten

Lijst met 2 elementen: [2, 3]
Element toevoegen: 1 : [2,3] = [1,2,3]
Lijsten samenvoegen: [1,2] ++ [3,4] = [1,2,3,4]
Lijst van karakters: ['a', 'b', 'c'] = 'abc'
Lengte van een lijst: length [1,2,3,4,5,6] = 6

Eerste element: head [1,2,3] = 1
Laatste element: last [1,2,3] = 3
Alles behalve eerste element: tail [1,2,3] = [2,3]
Alles behalve laatste element: init [1,2,3] = [1,2]

Enkele operaties:
Omdraaien: reverse [1,2,3] = [3,2,1]
N-de element: [1,2,3] !! 2 = 3 (Begin bij 0)
Leeg of niet: null [] = true, null [1,2,3] = false
Eerste n elementen: take 3 [1,2,3,4,5] = [1,2,3]

Tuples

Kan bestaan uit meerdere datatypen, (1,2) en (1, 'a') mag beide.
Een lijst van tuples mag alleen bestaan uit dezelfde tuples, [(1,2), (4,5)] mag, [(1,'a'), (2,'b')] mag ook, maar [(1,2), (3,'c')] mag niet.

3.2 Bash

NVT

3.3 Prolog

Kunnen er als volgt uit zien: [1,2,3,4,5]
Net als in haskell zijn head en tail te gebruiken op de volgende manier: [H|T] of [1|[2,3,4]]

Enkele ingebouwde manieren om met lijsten te werken:

- append\3:
  ?- append([1,2,3], [3,4,5], X).
  X = [1,2,4,5].
- member\2:
  ?- member(2, [1,2,3]).
  true.
- reverse\2:
  ?- reverse([1,2,3], X).
  X = [3,2,1].
- last\2:
  ?- last([1,2,3], 2).
  false.
- lenght\2:
  ?- length(X, 3).
  X = [_G927, _G930, _G933].
- nth0\3:
  ?- nth0(Y, [a,b,v], X).
  Y = 0,
  X = a;
  Y = 1,
  X = b;
  Y = 2,
  X = c.

3.4 Python

Tuples

Kunnen bestaan uit 1 of meerdere datatypen, net als bij haskell

Lists

Tussen blokhaken, net als bij haskell

Dictionaries

Zijn hetrogene hash tables (Java) die waarden opzoeken met behulp van een key, bijvoorbeeld:
myDict = 'ABC': 11
myDict = ['DEF'] = 22
myDict[33] = 'GHI'
print(myDict)

Dit geeft:
'DEF': 22, 33: 'GHI', 'ABC': 11

search = 'DEF'
if search in myDict:
—print(myDict[search])

Dit geeft:
22

Kijk voor de andere termen in de slides

3.5 Erlang

Lijsten

Homogene verzameling van elementen, homogeniteit is niet afgedwongen door Erlang
Over het algemeen vergelijkbaar met Prolog
Met ++ kunnen lijsten aan elkaar vast geplakt worden en met – kunnen ze van elkaar af worden getrokken. Beide gebeurt vanaf rechts naar links, bijvoorbeeld:
1> [1,2,3] ++ [4,5,6] = [1,2,3,4,5,6]
2> [1,2,3] – [1,2] – [3] = [3] (-> probeert eerst [3] van [1,2] af te halen, en dan [1,2] van [1,2,3])
3> ([1,2,3] – [1,2]) – [3] = []

3.6 Go

Een array kan als volgt gedeclareerd worden:
var a = [10]int

Als een array wordt meegegeven aan een functie dan wordt er eigenlijk een kopie meegegeven, als er iets in de functie wordt aangepast wordt dit dus niet in de originele array aangepast.
Als oplossing hierop gebruikt Go ook slices, deze worden gerepresenteerd door een onderliggende array en hebben geen vaste grootte (maximaal de grootte van de array).
Als er iets in een slice wordt aangepast in een functie dan overschrijft dit de array.

3.7 C++

Het is bij C++, net als bij C, belangrijk om gebruikt geheugen weer vrij te geven, dit gebeurd bij arrays op de volgende manier:

int main(){
—int* ip = new int[1000];
—(Gebruik hier het geheugen)
—delete[] ip;
}

4 Standaarddatatypen

4.1 Haskell

De standaarddatatypen zijn:

- Int: Heel getal, maar beperkt
- Integer: Heel getal, onbeperkt (langzamer)
- Float: Enkele pricisie float
- Double: Dubbele pricisie float
- Bool: True of False
- Char: Enkel karakter ([Char] is een String)

Het is ook mogelijk om eigen datatypen te maken, dit ziet bijvoorbeeld zo uit:
data Color = Red | Yellow | Blue | Green | Orange | Purple

In de terminal:
Prelude> let foo = Red
Prelude> :t (laat het datatype zien) foo -> Color

4.2 Bash

NVT

4.3 Prolog

NVT

4.4 Python

NVT

4.5 Erlang

De standaarddatatypen zijn:

- Numbers
- Variables (enkele keer aanmaken, beginnen met een hoofdletter)
- Atoms (beginnen met een kleine letter)
- Tuples
- Lists

4.6 Go

De standaarddatatypen zijn:

- bool
- string
- int
- uint
- float32, float64
- complex64, complex128
- Nog talloze andere integer-types elementen zoals de uint16

De standaardwaarde voor een aangemaakte variabele zonder waarde is 0, 0.0 of false

Met behulp van structs kunnen er andere datatypen aangemaakt worden, dit ziet er als volgt uit:

```
type Position struct {
—Row, Col int
}
    func main(){
—pos1 := Position{3,4}
—fmt.Println("Row:",pos1.Row, "Collumn:",pos1,Col )
}
```

Dit geeft Row: 3, Collumn: 4

4.7 C++

De standaarddatatypen zijn:

- bool
- char
- short
- int
- long
- float
- double