# PROGRAMMEREN

A constructor only works when you create a new instance of a class. This is the very first method to run on an instance, it has to run, and it runs exactly once.

A method on an instance can be called anywhere between zero times to infinite times on an instance once it is created.

A constructor is run implicitly. When a new instance of a class is created, it runs automatically. A method is run explicitly. It has to be called either from some outside source or from a method -or a constructor- in the class.

A constructor is intended to be used for wiring. In the constructor, you want to avoid doing actual work. You basically prepare the class to be used. A method is intended to do actual work.

public class MyType

{

private SomeType \_myNeeds;

// constructor

MyType(SomeType iWillNeedThis)

{

\_myNeeds = iWillNeedThis;

}

// method

public void MyMethod()

{

DoSomethingAbout(\_myNeeds);

}

}

*Wanneer iets een return type heeft zoals :void dan is het een functie/methode.*

*//Opdracht 1 (MACHTEN)ik geef m\* terug als antwoord en hij vult dit steeds in totdat hij n==0 tegenkomt en 1 teruggeeft vb: m=3 n=2, (1)m\* (2)m\* (3)1 omdat n dan 0 heeft bereikt (Heztelfde al m^2=m\*m).*

public int power(int m, int n) {

if(n==0){return 1;}

return m\* power(m,n-1);}

*//Opdracht 3(FACULTEIT) stopt wanneer 1 is bereikt.*

*// n =4, (1)4\* (2) 3\* (3) 2\* (4) 1 dus n\*(n-1)\*(n-2)\*(n-3)\*1*

public int factorial(int n) {

if (n == 1) return 1;

return n \* fac(n-1);}

## BINARY TREE

public class PersonTree {

private Person person;

private PersonTree lessEvil;

private PersonTree moreEvil;

public PersonTree(Person person) {

this.person = person;

lessEvil = null;

moreEvil = null;

}

//**Opdracht** **toevoegen van items**, links kleiner rechts groter (GEEN DUBBEL)

// **met new persontree(item)** geeft ik deze plek een nieuw middelpunt met een linker en rechter plaats.

//vanwege **this.person = person**, hoef ik in de methode geen this.person te gebruiken.

public void add(Person item) {

if(item.gtevilness() < Person.getevilness())

{

if(lessEvil == null)

{lessEvil = new PersonTree(item);}

else

{lessEvil.add(item);}

}

else

{

if(moreEvil == null)

{moreEvil = new PersonTree(item);}

else

{moreEvil.add(item);}

}

}

//**Opdracht count**

public int count() {

if(lessEvil == null)

{

if(moreEvil == null)

{return 1;}

else

{return 1 +moreEvil.count();} }

else

{if(moreEvil == null)

{return 1 +lessEvil.count();}

else {return 1 +moreEvil.count()+lessEvil.count();}

} }

# FUNCTIONEEL PROGRAMMEREN

vervang xs = [x\*y|x<-xs,y<-[1..3]

links staat wat je invult, x\*y is wat je terugkrijgt (in een lijst vanwege []) en naast de | staan de voorwaardes om iets terug te geven.

**Init:** geeft alles terug behalve het laatste getal

**Last:** geeft de laatste waarde terug

**Mod x y** == 0: is voor het kijken of er niks overblijft na het delen bvb 6/4 blijft 2 over.

**()** Een tuple is een soort van lijst. Echter mogen er in tuples elementen van verschillend typen staan.

**RETURN TYPE**

plus2 :: Double -> Double

plus2 x = x + 2.0

de eerste double is wat je erin gooit en de 2de double is wat voor soort antwoord.

**Int**

**Char**

**Bool**

**Float**

**Double**

**Reverse** [1,2,3] = [3,2,1]

**Integer**, een Int alleen dan zonder limiet

**Num**, een verzameling van alle numerieke datatypen

**Integral**, een verzameling van alle numerieke datatypen voor gehele getallen

**Fractional**, een verzameling van alle numerieke datatypen voor komma getallen

**Ord**, een verzameling van typen die geordend kunnen worden

**Eq**, een verzameling van typen die op gelijkheid getest kunnen worden

**=>** Alles ervoor heeft toepassing op alles erna.

**Elem**, zolang elem x in een lijst voorkomt geeft hij true terug vb: elem x [1x234] = TRUE.

**Quot =** quot ab = a/b en rond af naar 0 dus 3/2 =1 ipv van 1,5.

del :: Eq a => [a] -> [a] -> [a]

del xs ys = [y| y <- ys, not(elem y xs)]

vierkantBacktracking :: [(Int,Int,Int,Int,Int,Int,Int,Int,Int)]

vierkantBacktracking = [(x1, x2, x3, x4, x5, x6, x7, x8, x9) |

x1 <- [1..9],

x2 <- (del [x1] [1..9]),

x3 <- (del [x1,x2] [1..9]),

x1 + x2 + x3 == 15, -- de som van rij 1 is 15

x4 <- (del [x1,x2,x3] [1..9]),

x7 <- (del [x1,x2,x3,x4] [1..9]),

x1 + x4 + x7 == 15, -- de som van kolom 1 is 15

x5 <- (del [x1,x2,x3,x4,x7] [1..9]),

x3 + x5 + x7 == 15, -- de som van de diagonaal is 15

x6 <- (del [x1,x2,x3,x4,x5,x7] [1..9]),

x4 + x5 + x6 == 15, -- de som van rij 2 is 15

x9 <- (del [x1,x2,x3,x4,x5,x6,x7] [1..9]),

x3 + x6 + x9 == 15, -- de som van kolom 3 is 15

x1 + x5 + x9 == 15, -- de som van de diagonaal is 15

x8 <- (del [x1,x2,x3,x4,x5,x6,x7,x9] [1..9]),

x7 + x8 + x9 == 15, -- de som van rij 3 is 15

x2 + x5 + x8 == 15 -- de som van kolom 2 is 15]



