

Masterclass programmeren op de GR TI-84 (les 3)

*

Kevin van As

December 25, 2015

Recap!

We hebben gekeken naar:

- Datatype: Boolean *
- Controle middels condities: If
- Pseudocode & Algoritmes

Recap!

We hebben gekeken naar:

- Datatype: Boolean *
- Controle middels condities: If
- Pseudocode & Algoritmes

Recap!

We hebben gekeken naar:

- Datatype: Boolean *
- Controle middels condities: **If**
- Pseudocode & Algoritmes

Recap!

We hebben gekeken naar:

- Datatype: Boolean *
- Controle middels condities: **If**
- Pseudocode & Algoritmes

Vooruitzicht!

Vandaag zullen we kijken naar:

- Herhaling / Repetitie
 - Goto & Labels
 - For-loops
 - While-loops
- Pause
- Datatype: Lists

*

Vooruitzicht!

Vandaag zullen we kijken naar:

- Herhaling / Repetitie
 - Goto & Labels
 - For-loops
 - While-loops
- Pause
- Datatype: Lists

*

Vooruitzicht!

Vandaag zullen we kijken naar:

- Herhaling / Repetitie
 - Goto & Labels
 - For-loops
 - While-loops
- Pause
- Datatype: Lists

*

Vooruitzicht!

Vandaag zullen we kijken naar:

- Herhaling / Repetitie
 - Goto & Labels *
 - For-loops
 - While-loops
- Pause
- Datatype: Lists

Vooruitzicht!

Vandaag zullen we kijken naar:

- Herhaling / Repetitie
 - Goto & Labels *
 - For-loops
 - While-loops
- Pause
- Datatype: Lists

Vooruitzicht!

Vandaag zullen we kijken naar:

- Herhaling / Repetitie
 - Goto & Labels *
 - For-loops
 - While-loops
- Pause
- Datatype: Lists

Vooruitzicht!

Vandaag zullen we kijken naar:

- Herhaling / Repetitie
 - Goto & Labels
 - For-loops
 - While-loops
- Pause
- Datatype: Lists

*

Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*

R-r-r-repeat: Herhaling

- In een programma komt het vaak voor dat je een opdracht wilt herhalen.
- Kijk bijvoorbeeld naar `PROGRAM:WIEWINT` van de exercises van vorige les.
- Soms wil je een opdracht een gegeven aantal keer herhalen.
- Soms wil je een opdracht een van-te-voren onbekend aantal keer herhalen...
- Voor dit soort gevallen, bestaan er opdrachten die iets herhalen: `For`, `While` en `Goto` met `Lbl`.

R-r-r-repeat: Herhaling

- In een programma komt het vaak voor dat je een opdracht wilt herhalen.
- Kijk bijvoorbeeld naar **PROGRAM:WIEWINT** van de exercises van vorige les.
- Soms wil je een opdracht een gegeven aantal keer herhalen.
- Soms wil je een opdracht een van-te-voren onbekend aantal keer herhalen...
- Voor dit soort gevallen, bestaan er opdrachten die iets herhalen: For, While en Goto met Lbl.

R-r-r-repeat: Herhaling

- In een programma komt het vaak voor dat je een opdracht wilt herhalen.
- Kijk bijvoorbeeld naar **PROGRAM:WIEWINT** van de exercises van vorige les.
- Soms wil je een opdracht een ^{*}gegeven aantal keer herhalen.
- Soms wil je een opdracht een van-te-voren onbekend aantal keer herhalen...
- Voor dit soort gevallen, bestaan er opdrachten die iets herhalen: For, While en Goto met Lbl.

R-r-r-repeat: Herhaling

- In een programma komt het vaak voor dat je een opdracht wilt herhalen.
- Kijk bijvoorbeeld naar **PROGRAM:WIEWINT** van de exercises van vorige les.
- Soms wil je een opdracht een gegeven aantal keer herhalen. Dit is heel vaak hetzelfde programmeren...Stel je voor dat je iets 132x wilt doen!
- Soms wil je een opdracht een van-te-voren onbekend aantal keer herhalen...
- Voor dit soort gevallen, bestaan er opdrachten die iets herhalen: For, While en Goto met Lbl.

R-r-r-repeat: Herhaling

- In een programma komt het vaak voor dat je een opdracht wilt herhalen.
- Kijk bijvoorbeeld naar `PROGRAM:WIEWINT` van de exercises van vorige les.
- Soms wil je een opdracht een gegeven aantal keer herhalen. Dit is heel vaak hetzelfde programmeren...Stel je voor dat je iets 132x wilt doen!
- Soms wil je een opdracht een van-te-voren onbekend aantal keer herhalen...
- Voor dit soort gevallen, bestaan er opdrachten die iets herhalen: `For`, `While` en `Goto` met `Lbl`.

R-r-r-repeat: Herhaling

- In een programma komt het vaak voor dat je een opdracht wilt herhalen.
- Kijk bijvoorbeeld naar `PROGRAM:WIEWINT` van de exercises van vorige les.
- Soms wil je een opdracht een gegeven aantal keer herhalen. Dit is heel vaak hetzelfde programmeren...Stel je voor dat je iets 132x wilt doen!
- Soms wil je een opdracht een van-te-voren onbekend aantal keer herhalen...Dat is niet te programmeren...
- Voor dit soort gevallen, bestaan er opdrachten die iets herhalen: `For`, `While` en `Goto` met `Lbl`.

R-r-r-repeat: Herhaling

- In een programma komt het vaak voor dat je een opdracht wilt herhalen.
- Kijk bijvoorbeeld naar `PROGRAM:WIEWINT` van de exercises van vorige les.
- Soms wil je een opdracht een gegeven aantal keer herhalen. Dit is heel vaak hetzelfde programmeren...Stel je voor dat je iets 132x wilt doen!
- Soms wil je een opdracht een van-te-voren onbekend aantal keer herhalen...Dat is niet te programmeren...
- Voor dit soort gevallen, bestaan er opdrachten die iets herhalen: `For`, `While` en `Goto` met `Lbl`.


Het Lb1 (Label) statement

- Het Lb1 statement doet *niets* op zichzelf.
- Het markeert een locatie in het programma.
- Deze locatie kan elders in het programma gebruikt worden.
- Lb1 staat onder [PRGM] in het CTL-menu.
- Blader daarvoor naar beneden .

Het Lb1 (Label) statement

- Het Lb1 statement doet *niets* op zichzelf.
- Het markeert een locatie in het programma.
- Deze locatie kan elders in het programma gebruikt worden.
- Lb1 staat onder [PRGM] in het CTL-menu.
- Blader daarvoor naar beneden ▾).

Het Lb1 (Label) statement


- Het Lb1 statement doet *niets* op zichzelf.
- Het markeert een locatie in het programma.
- Deze locatie kan elders in het programma gebruikt worden.
- Lb1 staat onder [PRGM] in het CTL-menu.
- Blader daarvoor naar beneden .

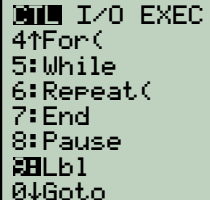
Het Lb1 (Label) statement

- Het Lb1 statement doet *niets* op zichzelf.
- Het markeert een locatie in het programma.
- Deze locatie kan elders in het programma gebruikt worden.
- Lb1 staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden ▾.

```
CTL I/O EXEC
LbIf
2:Then
3:Else
4:For(
5:While
6:Repeat(
7↓End
```


Het Lb1 (Label) statement

- Het Lb1 statement doet *niets* op zichzelf.
- Het markeert een locatie in het programma.
- Deze locatie kan elders in het programma gebruikt worden.
- Lb1 staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden .




```
I/O EXEC
4:For(
5:While
6:Repeat(
7:End
8:Pause
9:Lb1
0:Goto
```


Het Goto (Ga naar) statement

- Het **Goto** statement heeft **Lbl** nodig.
- Wanneer een programma bij de Goto-regel aan komt, dan springt hij naar de bijbehorende **Lbl** en gaat daar verder met het uitvoeren van het programma.*
- Je kunt zo dus door het hele programma heen springen!
- Goto staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden .

Het Goto (Ga naar) statement

- Het **Goto** statement heeft **Lb1** nodig.
- Wanneer een programma bij de **Goto**-regel aan komt, dan springt hij naar de bijbehorende **Lb1** en gaat daar verder met het uitvoeren van het programma.
- Je kunt zo dus door het hele programma heen springen!
- **Goto** staat onder **PRGM** in het **CTL**-menu.
- Blader daarvoor naar beneden .

Het Goto (Ga naar) statement


- Het **Goto** statement heeft **Lb1** nodig.
- Wanneer een programma bij de **Goto**-regel aan komt, dan springt hij naar de bijbehorende **Lb1** en gaat daar verder met het uitvoeren van het programma.
- Je kunt zo dus door het hele programma heen springen!
- **Goto** staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden .

Het Goto (Ga naar) statement

- Het **Goto** statement heeft **Lb1** nodig.
- Wanneer een programma bij de **Goto**-regel aan komt, dan springt hij naar de bijbehorende **Lb1** en gaat daar verder met het uitvoeren van het programma.
- Je kunt zo dus door het hele programma heen springen!
- **Goto** staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden ▾.

```
CTL I/O EXEC  
If  
2:Then  
3:Else  
4:For(  
5:While  
6:Repeat(  
7↓End
```

Het Goto (Ga naar) statement

- Het **Goto** statement heeft **Lb1** nodig.
- Wanneer een programma bij de **Goto**-regel aan komt, dan springt hij naar de bijbehorende **Lb1** en gaat daar verder met het uitvoeren van het programma.
- Je kunt zo dus door het hele programma heen springen!
- **Goto** staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden .

```
I/O EXEC  
4↑For(  
5:While  
6:Repeat(  
7:End  
8:Pause  
9:Lb1  
↵Goto
```

Het Goto (Ga naar) statement

Voorbeeld

- Voorbeeld: Het programma slaat hier

`:Disp "DIT WORDT NOOIT WEERGEVEN"` over en springt in één keer naar `:Disp "DIT WORDT ALTIJD UITGEVOERD"`.

Het negeert dus `Stop`: Dat wordt nooit uitgevoerd.

- Labels kunnen een naam hebben bestaande uit cijfers en letters, met maximaal 2 karakters.
 - Bijvoorbeeld AB of 0A, maar niet TUD2.

```
PROGRAM: GOTOLBL0
:Goto 0
:Disp "DIT WORDT
NOOIT WEERGEVEN"
:Stop
:Lbl 0
:Disp "DIT WORDT
ALTIJD
UITGEVOERD"
```

Het Goto (Ga naar) statement

Voorbeeld

- Voorbeeld: Het programma slaat hier

`:Disp "DIT WORDT NOOIT WEERGEVEN"` over en springt in één keer naar `:Disp "DIT WORDT ALTIJD UITGEVOERD"`.







Het negeert dus `Stop`: Dat wordt nooit uitgevoerd.

- Labels kunnen een naam hebben bestaande uit cijfers en letters, met maximaal 2 karakters.
 - Bijvoorbeeld AB of 0A, maar niet TUD2.

```
PROGRAM:GOTOLBL1
:Goto TU
:Disp "DIT WORDT
NOOIT WEERGEVEN"
:Stop
:Lb1 TU
:Disp "DIT WORDT
ALTIJD
UITGEVOERD"
```


Probeer Goto zelf!

Schrijf zelf een klein `PRGM` met alleen `Disp`, `Lbl` en `Goto` statements, om er een gevoel voor te krijgen.

CTL  EXEC	 I/O EXEC	 I/O EXEC
1:Input	4↑For(4↑For(
2:Prompt	5:While	5:While
 Disp	6:Repeat(6:Repeat(
4:DispGraph	7:End	7:End
5:DispTable	8:Pause	8:Pause
6:Output( Lbl	9:Lbl
7↓getKey	0↓Goto	 Goto

Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*

Wat is een For-loop?

- Alhoewel `Goto` en `Lbl` gebruikt kunnen worden om statements te herhalen, zijn er krachtigere methodes.
- Het `For` statement herhaalt een stukje code 'for (NI : voor)' bepaalde integers. *
- Herinner je dat 'integers' gehele getallen zijn
- Hiermee kun je een gegeven aantal keren een stukje code herhalen.
- Bijvoorbeeld: "Herhaal dit voor de integers 1 tot 5" = "Herhaal dit 5 keer"

Wat is een For-loop?

- Alhoewel **Goto** en **Lbl** gebruikt kunnen worden om statements te herhalen, zijn er krachtigere methodes.
- Het **For** statement herhaalt een stukje code 'for (NL: voor)' bepaalde integers. *
- Herinner je dat 'integers' gehele getallen zijn
- Hiermee kun je een gegeven aantal keren een stukje code herhalen.
- Bijvoorbeeld: "Herhaal dit voor de integers 1 tot 5" = "Herhaal dit 5 keer"

Wat is een For-loop?

- Alhoewel **Goto** en **Lbl** gebruikt kunnen worden om statements te herhalen, zijn er krachtigere methodes.
- Het **For** statement herhaalt een stukje code 'for (NL: voor)' bepaalde integers. *
- Herinner je dat 'integers' gehele getallen zijn
- Hiermee kun je een gegeven aantal keren een stukje code herhalen.
- Bijvoorbeeld: "Herhaal dit voor de integers 1 tot 5" = "Herhaal dit 5 keer"

Wat is een For-loop?

- Alhoewel **Goto** en **Lbl** gebruikt kunnen worden om statements te herhalen, zijn er krachtigere methodes.
- Het **For** statement herhaalt een stukje code 'for (NL: voor)' bepaalde integers. *
- Herinner je dat 'integers' gehele getallen zijn
- Hiermee kun je een gegeven aantal keren een stukje code herhalen.
- Bijvoorbeeld: "Herhaal dit voor de integers 1 tot 5" = "Herhaal dit 5 keer"

Wat is een For-loop?

- Alhoewel `Goto` en `Lbl` gebruikt kunnen worden om statements te herhalen, zijn er krachtigere methodes.
- Het `For` statement herhaalt een stukje code 'for (NL: voor)' bepaalde integers. *
- Herinner je dat 'integers' gehele getallen zijn
- Hiermee kun je een gegeven aantal keren een stukje code herhalen.
- Bijvoorbeeld: "Herhaal dit voor de integers 1 tot 5" = "Herhaal dit 5 keer"

Wat is een For-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Weergeef de integers 1 tot 5” = “Herhaal **Disp** 5 keer”
- Het is ook mogelijk getallen over te slaan:
 - “Weergeef de meervouden van 3 tot en met 9” = “Herhaal **Disp** <7> keer”
- De structuur is dus: **For**(variable,start,end,increment)
- **For** staat onder **PRGM** in het CTL-menu.
- De loop eindigt met de oude vertrouwde **End**

```
PROGRAM:FIRSTFOR  
:For(I,1,5)  
:Disp I  
:End
```


Wat is een For-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Weergeef de integers 1 tot 5” = “Herhaal **Disp** 5 keer”
- Het is ook mogelijk getallen over te slaan:
 - “Weergeef de meervouden van 3 tot en met 9” = “Herhaal **Disp** <??> keer”
- De structuur is dus: **For**(variable,start,end,increment)
- **For** staat onder **PRGM** in het CTL-menu.
- De loop eindigt met de oude vertrouwde **End**

```
PROGRAM:FIRSTFOR  
:For(I,3,9,3)  
:Disp I  
:End
```

Wat is een For-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Weergeef de integers 1 tot 5” = “Herhaal **Disp** 5 keer”
 - Het is ook mogelijk getallen over te slaan:
 - “Weergeef de meervouden van 3 tot en met 9” = “Herhaal **Disp** 3 keer”
- *
- De structuur is dus: **For**(variable,start,end,increment)
 - **For** staat onder **PRGM** in het CTL-menu.
 - De loop eindigt met de oude vertrouwde **End**

```
PROGRAM:FIRSTFOR
:For(I,3,9,3)
:Disp I
:End
```

Wat is een For-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Weergeef de integers 1 tot 5” = “Herhaal **Disp** 5 keer”
- Het is ook mogelijk getallen over te slaan:
 - “Weergeef de meervouden van 3 tot en met 9” = “Herhaal **Disp** 3 keer”
- De structuur is dus: **For**(variable,start,end,increment)
 - **For** staat onder **PRGM** in het CTL-menu.
 - De loop eindigt met de oude vertrouwde **End**

```
PROGRAM:FIRSTFOR
:For(I,3,9,3)
:Disp I
:End
```

Wat is een For-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Weergeef de integers 1 tot 5” = “Herhaal **Disp** 5 keer”
- Het is ook mogelijk getallen over te slaan:
 - “Weergeef de meervouden van 3 tot en met 9” = “Herhaal **Disp** 3 keer”
- De structuur is dus: **For**(variable,start,end,increment)
- **For**(staat onder **PRGM** in het CTL-menu.
- De loop eindigt met de oude vertrouwde **End**

```
CTL I/O EXEC
1:If
2:Then
3:Else
4:For(
5:While
6:Repeat(
7:End
```

Wat is een For-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Weergeef de integers 1 tot 5” = “Herhaal **Disp** 5 keer”
- Het is ook mogelijk getallen over te slaan:
 - “Weergeef de meervouden van 3 tot en met 9” = “Herhaal **Disp** 3 keer”
- De structuur is dus: **For**(variable,start,end,increment)
- **For**(staat onder **PRGM** in het CTL-menu.
- De loop eindigt met de oude vertrouwde **End**

```
PROGRAM:FIRSTFOR  
:For(I,1,5)  
:Disp I  
:End
```

For< probeer het zelf!

Probeer de volgende Programs te schrijven:

- 1 **DisP** de getallen 1, 2, 3, 4, 5
- 2 **DisP** de getallen -1, 0, 1
- 3 **DisP** de getallen 1, 0, -1 (in die volgorde!)
- 4 **DisP** de getallen 1, 2, 3, 4, 5, *11, 12, 13, 14, 15
- 5 **DisP** alle positieve even getallen tot en met 12.
- 6 **DisP** alle positieve oneven getallen tot en met 11.

```
PROGRAM:FIRSTFOR  
:For(I,3,9,3)  
:DisP I  
:End
```

```
I/O EXEC  
1:If  
2:Then  
3:Else  
4:For<  
5:While  
6:Repeat<  
7↓End
```

For< probeer het zelf!

Een paar antwoorden

- 1 **Disp** de getallen 1, 0, -1 (in die volgorde!)
- 2 **Disp** de getallen 1, 2, 3, 4, 5, 11, 12, 13, 14, 15
- 3 **Disp** alle positieve oneven getallen tot en met 11.

```
PROGRAM:FORMIN1  
:For(I,1,-1,-1)  
:Disp I  
:End
```

```
PROGRAM:FORODD  
:For(I,1,11,2)  
:Disp I  
:End
```

```
PROGRAM:FORDUBLE  
:For(I,1,5)  
:Disp I  
:End  
:For(I,11,15)  
:Disp I  
:End
```

Of als de volgorde
niet uitmaakt:

```
PROGRAM:FORDBLE2  
:For(I,1,5)  
:Disp I  
:Disp I+10  
:End
```

Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*

Wat is een while-loop?

- **For** is leuk indien je weet hoe vaak je iets wilt herhalen.
- **While** is leuk indien je dat à priori juist *niet* weet.
- Een **while**-loop gaat door totdat een conditie (**If**) onwaar wordt.*
- Bijvoorbeeld:
 - "Herhaal dit totdat het regent"
 - "Herhaal dit totdat de berekening nauwkeurig genoeg is"

Wat is een while-loop?

- **For** is leuk indien je weet hoe vaak je iets wilt herhalen.
- **While** is leuk indien je dat à priori juist *niet* weet.
- Een **while**-loop gaat door totdat een conditie (If) onwaar wordt.*
- Bijvoorbeeld:
 - "Herhaal dit totdat het regent"
 - "Herhaal dit totdat de berekening nauwkeurig genoeg is"

Wat is een while-loop?

- For is leuk indien je weet hoe vaak je iets wilt herhalen.
- While is leuk indien je dat à priori juist *niet* weet.
- Een While-loop gaat door totdat een conditie (If) onwaar wordt.
- Bijvoorbeeld:
 - "Herhaal dit totdat het regent"
 - "Herhaal dit totdat de berekening nauwkeurig genoeg is"

Wat is een while-loop?

- **For** is leuk indien je weet hoe vaak je iets wilt herhalen.
- **While** is leuk indien je dat à priori juist *niet* weet.
- Een **While**-loop gaat door totdat een conditie (**If**) onwaar wordt.*
- Bijvoorbeeld:
 - "Herhaal dit totdat het regent"
 - "Herhaal dit totdat de berekening nauwkeurig genoeg is"

Wat is een while-loop?

- **For** is leuk indien je weet hoe vaak je iets wilt herhalen.
- **While** is leuk indien je dat à priori juist *niet* weet.
- Een **While**-loop gaat door totdat een conditie (**If**) onwaar wordt.*
- Bijvoorbeeld:
 - “Herhaal dit totdat het regent”
 - “Herhaal dit totdat de berekening nauwkeurig genoeg is”

Wat is een `while`-loop?

- `For` is leuk indien je weet hoe vaak je iets wilt herhalen.
- `While` is leuk indien je dat à priori juist *niet* weet.
- Een `While`-loop gaat door totdat een conditie (`If`) onwaar wordt.
- Bijvoorbeeld:
 - “Herhaal dit totdat het regent”
 - “Herhaal dit totdat de berekening nauwkeurig genoeg is”

Wat is een while-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Herhaal dit totdat het regent”
 - `int(rand*10)` genereert een willekeurige integer tussen 0 en 9. Dit geeft een 10% kans dat de loop breaks (=stopt) elke keer dat hij execute (=herhaalt).
- De structuur is dus: `While` * «conditie»
- `While` staat onder `PRGM` in het CTL-menu.
- De loop eindigt met de oude vertrouwde `End`

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
:int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

Wat is een while-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Herhaal dit totdat het regent”
 - `int(rand*10)` genereert een willekeurige integer tussen 0 en 9. Dit geeft een 10% kans dat de loop breaks (=stopt) elke keer dat hij execute (=herhaalt).
- De structuur is dus: `While` * «conditie»
- `While` staat onder `PRGM` in het CTL-menu.
- De loop eindigt met de oude vertrouwde `End`

```
DROOG  
DROOG  
DROOG  
DROOG  
REGEN
```

Done

```
PROGRAM:FRSTWHLE  
:0→R:While R≠1  
:int(rand*10)→R  
:Disp "DROOG"  
:End  
:Disp "REGEN"
```


Wat is een while-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Herhaal dit totdat het regent”
 - `int(rand*10)` genereert een willekeurige integer tussen 0 en 9. Dit geeft een 10% kans dat de loop breaks (=stopt) elke keer dat hij execute (=herhaalt).
- De structuur is dus: `While` * «conditie»
- `While` staat onder `PRGM` in het CTL-menu.
- De loop eindigt met de oude vertrouwde `End`

```
DROOG  
DROOG  
DROOG  
DROOG  
REGEN
```

Done

```
PROGRAM:FRSTWHLE  
:0→R:While R≠1  
:int(rand*10)→R  
:Disp "DROOG"  
:End  
:Disp "REGEN"
```

Wat is een while-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Herhaal dit totdat het regent”
 - `int(rand*10)` genereert een willekeurige integer tussen 0 en 9. Dit geeft een 10% kans dat de loop breaks (=stopt) elke keer dat hij execute (=herhaalt).
- De structuur is dus: **While** * «conditie»
- **While** staat onder **PRGM** in het CTL-menu.
- De loop eindigt met de oude vertrouwde **End**

```
DROOG  
DROOG  
DROOG  
DROOG  
REGEN
```

Done

```
PROGRAM:FRSTWHLE  
:0→R:While R≠1  
:int(rand*10)→R  
:Disp "DROOG"  
:End  
:Disp "REGEN"
```

Wat is een while-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Herhaal dit totdat het regent”
 - `int(rand*10)` genereert een willekeurige integer tussen 0 en 9. Dit geeft een 10% kans dat de loop breaks (=stopt) elke keer dat hij execute (=herhaalt).
- De structuur is dus: **While** * «conditie»
- **While** staat onder **PRGM** in het CTL-menu.
- De loop eindigt met de oude vertrouwde End

```
DROOG  
DROOG  
DROOG  
DROOG  
REGEN
```

Done

```
CTL I/O EXEC  
1: If  
2: Then  
3: Else  
4: For(  
5: While  
6: Repeat(  
7: End
```

Wat is een while-loop?

- Op de rekenmachine ziet dit er zo uit:
 - “Herhaal dit totdat het regent”
 - `int(rand*10)` genereert een willekeurige integer tussen 0 en 9. Dit geeft een 10% kans dat de loop breaks (=stopt) elke keer dat hij execute (=herhaalt).
- De structuur is dus: **While** * «conditie»
- **While** staat onder **PRGM** in het CTL-menu.
- De loop eindigt met de oude vertrouwde **End**

```
DROOG  
DROOG  
DROOG  
DROOG  
REGEN
```

Done

```
PROGRAM:FRSTWHLE  
:0→R:While R≠1  
:int(rand*10)→R  
:Disp "DROOG"  
:END  
:Disp "REGEN"
```

While zelf aan de slag End

- ① Maak `FRSTWHLE` na. `int()` en `rand` staan onder `MATH`.
- ② Voer dit `prog` een aantal keer uit. Zie je altijd hetzelfde?
- ③ Hoe kun je het verwachte aantal repetities veranderen?
- ④ Breid het programma nu uit met de zon: Pas als de zon stopt met schijnen, dan kan het gaan regenen.
 - Tip: Dit betekent dat je een (bijna) identieke `While`-loop nodig hebt voor de zon, binnenin de loop voor de regen.

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
MATH NUM CPX PRB
1:rand(
2:nPr
3:nCr
4:!
5:randInt(
6:randNorm(
7:randBin(
```

```
MATH NUM CPX PRB
1:abs(
2:round(
3:iPart(
4:fPart(
5:int(
6:min(
7:max(
```


While zelf aan de slag End

Antwoord

- 1 Voer dit `program` een aantal keer uit. Zie je altijd hetzelfde?
 - Hoe vaak de loop wordt uitgevoerd varieert!
- 2 Hoe kun je het verwachte aantal repetities veranderen?
 - Het getal '`10`' moet verandert worden. Groter = meer repetities.
- 3 Breid het programma nu uit met de zon: Pas als de zon stopt met schijnen, dan kan het gaan regenen.

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*(10))→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
DROOG
DROOG
DROOG
DROOG
DROOG
DROOG
DROOG
REGEN
```

Done

While zelf aan de slag End

Antwoord

- ① Voer dit ~~prog~~ een aantal keer uit. Zie je altijd hetzelfde?
 - Hoe vaak de loop wordt uitgevoerd varieert!
- ② Hoe kun je het verwachte aantal repetities veranderen?
 - Het getal '10' moet verandert worden. Groter = meer repetities. *
- ③ Breid het programma nu uit met de zon: Pas als de zon stopt met schijnen, dan kan het gaan regenen.

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```


```
PROGRAM:SCNDWHLE
:0→R:While R≠1
:0→S:While S≠1
: int(rand*4)→S
:Disp "ZON"
:End
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```


Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*


Intermezzo: Pause

- In de **While-Prgrms** van de vorige slides, werd je beeldscherm behoorlijk vol geschreven.
- Zo kan de gebruiker natuurlijk niet zien wat we **Disp**-en!
- Met **Pause** kun je de executie van het programma pauseren.
- Executie gaat verder wanneer de gebruiker op **[ENTER]** drukt.
- **Pause** staat onder **[PRGM]** in het CTL-menu.
- Blader daarvoor naar beneden .
- Speel ermee door **Pause** toe te voegen in **SCNDWHLE!**

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
PROGRAM:SCNDWHLE
:0→R:While R≠1
:0→S:While S≠1
: int(rand*4)→S
:Disp "ZON"
:End
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```


Intermezzo: Pause

- In de **While-Prgrms** van de vorige slides, werd je beeldscherm behoorlijk vol geschreven.
- Zo kan de gebruiker natuurlijk niet zien wat we **Disp**-en!
- Met **Pause** kun je de executie van het programma pauzeren.
- Executie gaat verder wanneer de gebruiker op **[ENTER]** drukt.
- **Pause** staat onder **[PRGM]** in het CTL-menu.
- Blader daarvoor naar beneden .
- Speel ermee door **Pause** toe te voegen in **SCNDWHLE!**

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
PROGRAM:SCNDWHLE
:0→R:While R≠1
:0→S:While S≠1
: int(rand*4)→S
:Disp "ZON"
:End
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```


Intermezzo: Pause

- In de **While-Prgrms** van de vorige slides, werd je beeldscherm behoorlijk vol geschreven.
- Zo kan de gebruiker natuurlijk niet zien wat we **Disp**-en!
- Met **Pause** kun je de executie van het programma pauseren.
- Executie gaat verder wanneer de gebruiker op **ENTER** drukt.
- **Pause** staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden .
- Speel ermee door **Pause** toe te voegen in **SCNDWHLE**!

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
PROGRAM:SCNDWHLE
:0→R:While R≠1
:0→S:While S≠1
: int(rand*4)→S
:Disp "ZON"
:End
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

Intermezzo: Pause

- In de **While-Prgrms** van de vorige slides, werd je beeldscherm behoorlijk vol geschreven.
- Zo kan de gebruiker natuurlijk niet zien wat we **Disp**-en!
- Met **Pause** kun je de executie van het programma pauseren.
- Executie gaat verder wanneer de gebruiker op **ENTER** drukt.
- **Pause** staat onder **[PRGM]** in het CTL-menu.
- Blader daarvoor naar beneden .
- Speel ermee door **Pause** toe te voegen in **SCNDWHLE!**

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
PROGRAM:SCNDWHLE
:0→R:While R≠1
:0→S:While S≠1
: int(rand*4)→S
:Disp "ZON"
:End
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```


Intermezzo: Pause

- In de **While-Prgrms** van de vorige slides, werd je beeldscherm behoorlijk vol geschreven.
- Zo kan de gebruiker natuurlijk niet zien wat we **Disp**en!
- Met **Pause** kun je de executie van het programma pauseren.
- Executie gaat verder wanneer de gebruiker op **ENTER** drukt.
- **Pause** staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden ▾.
- Speel ermee door **Pause** toe te voegen in **SCNDWHLE!**

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
CTL I/O EXEC
If
2:Then
3:Else
4:For(
5:While
6:Repeat(
7↓End
```


Intermezzo: Pause

- In de **While-Prgrms** van de vorige slides, werd je beeldscherm behoorlijk vol geschreven.
- Zo kan de gebruiker natuurlijk niet zien wat we **DisP**en!
- Met **Pause** kun je de executie van het programma pauseren.
- Executie gaat verder wanneer de gebruiker op **ENTER** drukt.
- **Pause** staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden .
- Speel ermee door **Pause** toe te voegen in **SCNDWHLE!**

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
: DisP "DROOG"
: End
: DisP "REGEN"
```

```
CTL I/O EXEC
4↑For(
5:While
6:Repeat(
7:End
8↓Pause
9:Lbl
0↓Goto
```

Intermezzo: Pause

- In de **While-Prgrms** van de vorige slides, werd je beeldscherm behoorlijk vol geschreven.
- Zo kan de gebruiker natuurlijk niet zien wat we **Disp**-en!
- Met **Pause** kun je de executie van het programma pauzeren.
- Executie gaat verder wanneer de gebruiker op **ENTER** drukt.
- **Pause** staat onder **PRGM** in het CTL-menu.
- Blader daarvoor naar beneden .
- Speel ermee door **Pause** toe te voegen in **SCNDWHLE**!

```
PROGRAM:FRSTWHLE
:0→R:While R≠1
: int(rand*10)→R
:Disp "DROOG"
:End
:Disp "REGEN"
```

```
PROGRAM:SCNDWHLE
:0→R:While R≠1
:0→S:While S≠1
: int(rand*4)→S
:Disp "ZON"
:End
: int(rand*10)→R
:Disp "DROOG"
:Pause
:End
:Disp "REGEN"
```


Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*

Een nieuw datatype: List

- Wat is beter dan één getal?
 - Een list (NL: lijst) kan meerdere getallen in dezelfde variabele storen.
 - Dit maakt het makkelijk om over een lijst getallen te itereren (≡lopen).

Een nieuw datatype: List

- Wat is beter dan één getal?
 - MEER GETALLEN!
- Een list (NL: lijst) kan meerdere getallen in dezelfde variabele storen. *
- Dit maakt het makkelijk om over een lijst getallen te itereren (≡lopen).

Een nieuw datatype: List

- Wat is beter dan één getal?
 - MEER GETALLEN!
- Een list (NL: lijst) kan meerdere ^{*}getallen in dezelfde variabele storen.
- Dit maakt het makkelijk om over een lijst getallen te itereren (≡lopen).

Een nieuw datatype: List

- Wat is beter dan één getal?
 - MEER GETALLEN!
- Een list (NL: lijst) kan meerdere ^{*}getallen in dezelfde variabele storen.
- Dit maakt het makkelijk om over een lijst getallen te itereren (=lopen).

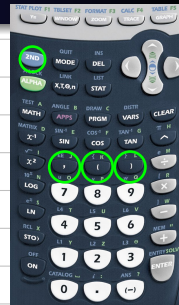
Een nieuw datatype: List

- Een lijst store je in de L_1 to L_6 variables.
- Op het scherm van de rekenmachine definieer je een list tussen accolades ({}). ^{*} Probeer het zelf!
- Indien je het derde getal uit list L_1 wilt hebben, dan type je $L_1(3)$.



Een nieuw datatype: List

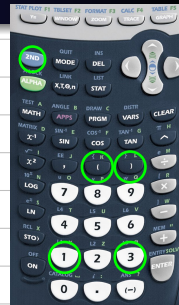
- Een lijst store je in de L_1 to L_6 variables.
- Op het scherm van de rekenmachine definieer je een list tussen accolades ({}). Probeer het zelf!
- Indien je het derde getal uit list L_1 wilt hebben, dan type je $L_1(3)$.



```
{5,3,2}→L1
                    {5,3,2}
L1
                    {5,3,2}
```

Een nieuw datatype: List

- Een lijst store je in de L_1 to L_6 variables.
- Op het scherm van de rekenmachine definieer je een list tussen accolades ({}).^{*} Probeer het zelf!
- Indien je het derde getal uit list L_1 wilt hebben, dan type je $L_1(3)$.



```
(5,3,2)→L1
                    (5,3,2)
L1
                    (5,3,2)
L1(3)
                    2
```

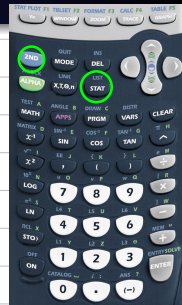

Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 **Datatype: Lists**
 - List datatype
 - **List hulpjes**
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*

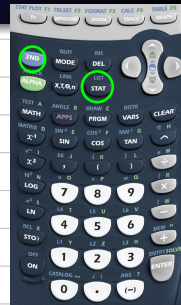
Nuttige hulpjes voor Lists

- Onder `2nd` `STAT` = `[LIST]` staan nuttige hulpjes voor Lists
- Onder `NAMES` staan variabelen die je als List kunt gebruiken. Dit zijn de vertrouwde `L1` to `L6`, maar dan nog veel meer mogelijkheden.*
- Onder `DPS` staan list-operaties, zoals sorteren (`SortA()`), of het aantal elementen in een lijst opvragen (`dim()`).
- Onder `MATH` staan wiskundige operaties, zoals het berekenen van het gemiddelde (`mean()`).



Nuttige hulpjes voor Lists

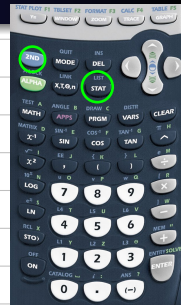
- Onder **2nd** **STAT** = **[LIST]** staan nuttige hulpjes voor Lists
- Onder **NAMES** staan variabelen die je als List kunt gebruiken. Dit zijn de vertrouwde **L₁** to **L₆**, maar dan nog veel meer mogelijkheden.*
- Onder **OPS** staan list-operaties, zoals sorteren (**SortA()**), of het aantal elementen in een lijst opvragen (**dim()**).
- Onder **MATH** staan wiskundige operaties, zoals het berekenen van het gemiddelde (**mean()**).



```
NAMES OPS MATH
1:L1
2:L2
3:L3
4:L4
5:L5
6:L6
7:↓A
```

Nuttige hulpjes voor Lists

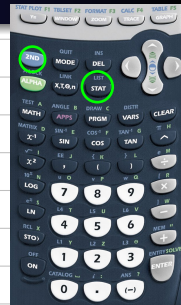
- Onder **2nd** **STAT** = **[LIST]** staan nuttige hulpjes voor Lists
- Onder **NAMES** staan variabelen die je als List kunt gebruiken. Dit zijn de vertrouwde **L₁** to **L₆**, maar dan nog veel meer mogelijkheden.*
- Onder **OPS** staan list-operaties, zoals sorteren (**SortA()**), of het aantal elementen in een lijst opvragen (**dim()**).
- Onder **MATH** staan wiskundige operaties, zoals het berekenen van het gemiddelde (**mean()**).



```
NAMES OPS MATH
1:SortA(
2:SortD(
3:dim(
4:Fill(
5:seq(
6:cumSum(
7:↓List(
```

Nuttige hulpjes voor Lists

- Onder `2nd` `STAT` = `[LIST]` staan nuttige hulpjes voor Lists
- Onder `NAMES` staan variabelen die je als List kunt gebruiken. Dit zijn de vertrouwde `L1` to `L6`, maar dan nog veel meer mogelijkheden.*
- Onder `OPS` staan list-operaties, zoals sorteren (`SortA()`), of het aantal elementen in een lijst opvragen (`dim()`).
- Onder `MATH` staan wiskundige operaties, zoals het berekenen van het gemiddelde (`mean()`).



```
NAMES OPS MATH
1:min(
2:max(
3:mean(
4:median(
5:sum(
6:prod(
7:stdDev(
```

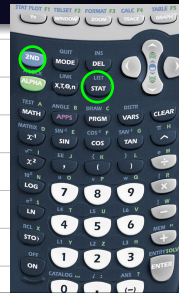
Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA()`?
- Wat doet `SortD()`?
- Wat doet `max()`?
- Wat doet `prod()`?
- Wat doet `seq()`?
- Wat doet `ΔList`?

*

- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!



```
(5,2,3,0)→L1  
      (5 2 3 0)
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA`(?)

- Wat doet `SortD`(?)

- Wat doet `max`(?)

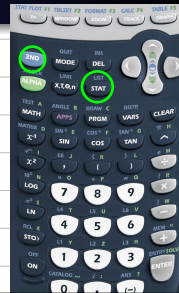
- Wat doet `prod`(?)

*

- Wat doet `seq`(?)

- Wat doet `ΔList`?

- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

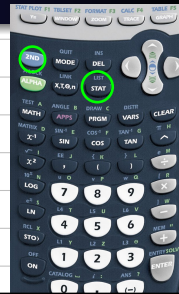


```
(5,2,3,0)→L1
      (5 2 3 0)
SortA(L1)
Done
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?` Sorteert oplopend (ascending)!
- Wat doet `SortD(?`
- Wat doet `max(?`
- Wat doet `prod(?`
- Wat doet `seq(?`
- Wat doet `ΔList?`
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

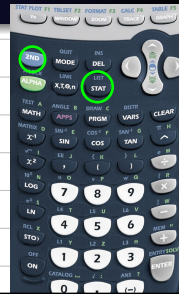


```
(5,2,3,0)→L1
      (5 2 3 0)
SortA(L1)
      Done
L1
      (0 2 3 5)
```


Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)`
- Wat doet `max(?)`
- Wat doet `prod(?)`
- Wat doet `seq(?)`
- Wat doet `ΔList?`
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

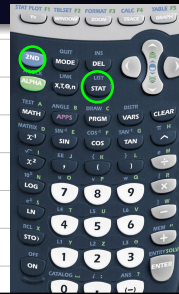


```
(5,2,3,0)→L1
          (5 2 3 0)
SortA(L1)
          Done
L1
          (0 2 3 5)
SortD(L1)
          Done
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?` Sorteert oplopend (ascending)!
- Wat doet `SortD(?` Sorteert aflopend (descending)!
- Wat doet `max(?`
- Wat doet `prod(?`
- Wat doet `seq(?`
- Wat doet `ΔList?`
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!



```
(5,2,3,0)→L1
      (5 2 3 0)
SortA(L1)
      Done
L1
      (0 2 3 5)
SortD(L1)
      Done
L1
      (5 3 2 0)
```

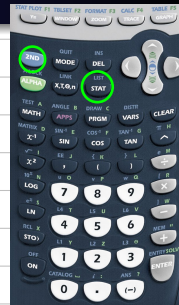
Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteer oplopend (ascending)!
- Wat doet `SortD(?)` Sorteer aflopend (descending)!
- Wat doet `max(?)`
- Wat doet `prod(?)`
- Wat doet `seq(?)`
- Wat doet `ΔList?`

*

- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

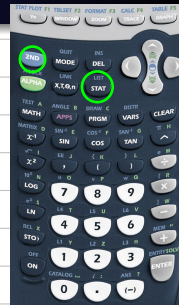


```
(5,2,3,0)→L1
(5 2 3 0)
max(L1)
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)` Sorteert aflopend (descending)!
- Wat doet `max(?)` Maximale waarde!
- Wat doet `prod(?)`
- Wat doet `seq(?)`
- Wat doet `ΔList?`
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!



```
(5,2,3,0)→L1
(5 2 3 0)
max(L1)
5
```

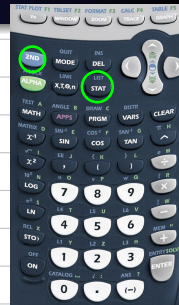
Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)` Sorteert aflopend (descending)!
- Wat doet `max(?)` Maximale waarde!
- Wat doet `prod(?)`

- Wat doet `seq(?)`
- Wat doet `ΔList?`

- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

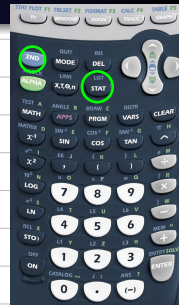


```
(5,2,3,0)→L1
      (5 2 3 0)
max(L1)
      5
prod(L1,1,3)
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)` Sorteert aflopend (descending)!
- Wat doet `max(?)` Maximale waarde!
- Wat doet `Prod(?)` Vermenigvuldigt alle getallen tussen index 1 en 3!
- Wat doet `seq(?)`
- Wat doet `ΔList?`
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!



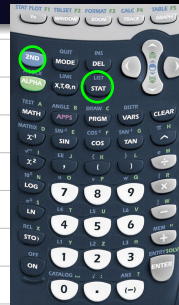
```
(5,2,3,0)→L1
(5 2 3 0)
max(L1)
5
Prod(L1,1,3)
30
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)` Sorteert aflopend (descending)!
- Wat doet `max(?)` Maximale waarde!
- Wat doet `prod(?)` Vermenigvuldigt alle getallen tussen index 1 en 3!
- Wat doet `seq(?)`
- Wat doet `ΔList?`

- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

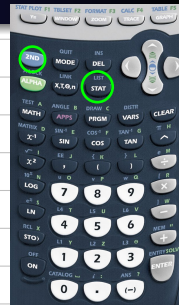


```
seq(X²,X,0,5,1)→L₁  
ΔList(L₁)
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)` Sorteert aflopend (descending)!
- Wat doet `max(?)` Maximale waarde!
- Wat doet `prod(?)` Vermenigvuldigt alle getallen tussen index 1 en 3!
- Wat doet `seq(?)` Het vult een list m.b.v. een formule!
- Wat doet `ΔList?`



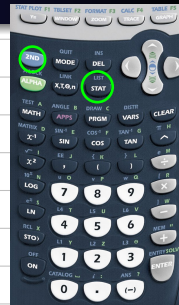
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

```
seq(X^2,X,0,5,1)→L1
  {0 1 4 9 16 25}
ΔList(L1)
```


Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)` Sorteert aflopend (descending)!
- Wat doet `max(?)` Maximale waarde!
- Wat doet `prod(?)` Vermenigvuldigt alle getallen tussen index 1 en 3!
- Wat doet `seq(?)` Het vult een list m.b.v. een formule!
- Wat doet `ΔList?`



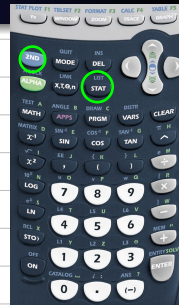
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

```
seq(X^2,X,0,5,1)→L1
  {0 1 4 9 16 25}
ΔList(L1)
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?` Sorteert oplopend (ascending)!
- Wat doet `SortD(?` Sorteert aflopend (descending)!
- Wat doet `max(?` Maximale waarde!
- Wat doet `Prod(?` Vermenigvuldigt alle getallen tussen index 1 en 3!
- Wat doet `seq(?` Het vult een list m.b.v. een formule!
- Wat doet `ΔList?` Het geeft een list die één korter is en het verschil tussen opvolgende elementen heeft.
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!

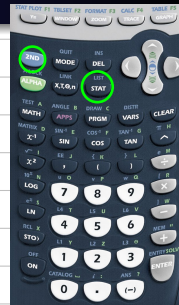


```
seq(X^2,X,0,5,1)→L1
(0 1 4 9 16 25)
ΔList(L1)
(1 3 5 7 9)
```

Nuttige hulpjes voor Lists

Laten we wat operaties uit proberen:

- Wat doet `SortA(?)` Sorteert oplopend (ascending)!
- Wat doet `SortD(?)` Sorteert aflopend (descending)!
- Wat doet `max(?)` Maximale waarde!
- Wat doet `Prod(?)` Vermenigvuldigt alle getallen tussen index 1 en 3!
- Wat doet `seq(?)` Het vult een list m.b.v. een formule!
- Wat doet `ΔList?` Het geeft een list die één korter is en het verschil tussen opvolgende elementen heeft.
- Dit is allemaal een stuk krachtiger dan rekenen met een enkel getal!



```
seq(X^2,X,0,5,1)→L1
(0 1 4 9 16 25)
ΔList(L1)
(1 3 5 7 9)
```

WIEWINT met Lists

Kun je je dit programma nog herinneren? Nasty...

```
PROGRAM:WIEWINT
:Prompt A
:Prompt B
:Prompt C
:Prompt D
:
:If A>B:Then
:If A>C:Then
:If A>D:Then
:Disp "A WINS"
:Else
:Disp "D WINS"
:End
:Else
:If C>D:Then
:Disp "C WINS"
```

*

```
:Else
:Disp "D WINS"
:End
:End
:Else
:If B>C:Then
:If B>D:Then
:Disp "B WINS"
:Else
:Disp "D WINS"
:End
:Else
:If C>D:Then
:Disp "C WINS"
:Else
:Disp "D WINS"
:End
:End
:End
```

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

1: **function** WIEWINT(list met scores)

*

8: **end function**

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

- 1: **function** WIEWINT(list met scores)
- 2: Vindt punten van de winnaar: M

*

8: **end function**

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

```
1: function WIEWINT(list met scores)
2:   Vindt punten van de winnaar: M
3:   for alle spelers do *
```



```
7:   end for
8: end function
```

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

```
1: function WIEWINT(list met scores)
2:   Vindt punten van de winnaar: M
3:   for alle spelers do *
4:     if punten = M then

6:     end if
7:   end for
8: end function
```

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

```
1: function WIEWINT(list met scores)
2:   Vindt punten van de winnaar: M
3:   for alle spelers do *
4:     if punten = M then
5:       Deze speler wint!
6:     end if
7:   end for
8: end function
```

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

```
1: function WIEWINT(list met scores)
2:   Vindt punten van de winnaar: M
3:   for alle spelers do *
4:     if punten = M then
5:       Deze speler wint!
6:     end if
7:   end for
8: end function
```

```
PROGRAM: WIEWINT2
: Prompt L1
: max(L1) → M
: Disp "WINNERS: "
: For(I, 1, dim(L1))
:   If M=L1(I): Then
:     Disp I
:   End
: End
```

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

```
1: function WIEWINT(list met scores)
2:   Vindt punten van de winnaar: M
3:   for alle spelers do *
4:     if punten = M then
5:       Deze speler wint!
6:     end if
7:   end for
8: end function
```

```
PROGRAM:WIEWINT2
:Prompt L1
:max(L1)→M
:Disp "WINNERS:"
:For(I,1,dim(L1))
:If M=L1(I):Then
:Disp I
:End
:End
```

Van 33 regels voor 4 spelers, naar slechts 8 regels voor een **onbeperkt** aantal spelers! Kun je de kracht van loops en lists al waarderen?

WIEWINT met Lists

Algorithm 1 “WieWint? met Lists”

```

1: function WIEWINT(list met scores)
2:     Vindt punten van de winnaar: M
3:     for alle spelers do *
4:         if punten = M then
5:             Deze speler wint!
6:         end if
7:     end for
8: end function
    
```

```

PROGRAM:WIEWINT2
:Prompt L1
:max(L1)→M
:Disp "WINNERS:"
:For(I,1,dim(L1))
:If M=L1(I):Then
:Disp I
:End
:End
    
```

```

PrgrmWIEWINT2
L1=?{5,3,5,2
WINNERS:
                                     1
                                     3
                                     Done
    
```

Van 33 regels voor 4 spelers, naar slechts 8 regels voor een **onbeperkt** aantal spelers! Kun je de kracht van loops en lists al waarderen?

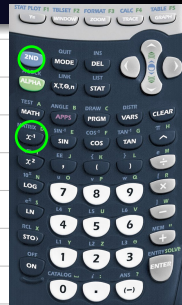
Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - **Matrix datatype**
- 4 Exercises
 - Exercises
 - Answers

*

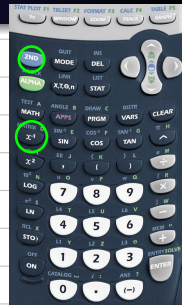
Sidenote: Matrix datatype

- Een List is een één-dimensionale lijst van getallen
- Zonder in details te treden, een Matrix is een twee-dimensionale lijst van getallen.
- Matrices vind je onder 2^{nd} X^{-1} = [MATRIX]
- Dit kan nuttig zijn voor je data opslag



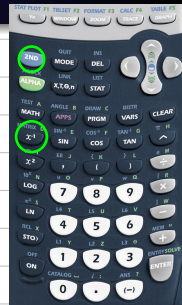
Sidenote: Matrix datatype

- Een List is een één-dimensionale lijst van getallen
- Zonder in details te treden, * een Matrix is een twee-dimensionale lijst van getallen.
- Matrices vind je onder 2^{nd} x^{-1} = [MATRIX]
- Dit kan nuttig zijn voor je data opslag



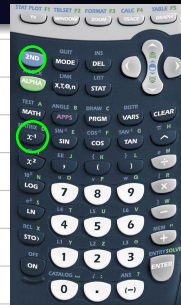
Sidenote: Matrix datatype

- Een List is een één-dimensionale lijst van getallen
- Zonder in details te treden, * een Matrix is een twee-dimensionale lijst van getallen.
- Matrices vind je onder $\boxed{2nd} \boxed{x^{-1}} = \boxed{[MATRIX]}$
- Dit kan nuttig zijn voor je data opslag



Sidenote: Matrix datatype

- Een List is een één-dimensionale lijst van getallen
- Zonder in details te treden, * een Matrix is een twee-dimensionale lijst van getallen.
- Matrices vind je onder $\boxed{2nd} \boxed{x^{-1}} = \boxed{[MATRIX]}$
- Dit kan nuttig zijn voor je data opslag.



Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*

WIEWINT3: Wie worden er tweede en derde?

Breid WIEWINT2 uit door ook te weergeven wie tweede en derde zijn geworden.

Tip: Er bestaat geen “max2” functie of iets dergelijks voor het vinden van het op-één-na maximum, dus je zult iets anders moeten bedenken met behulp van `max`, of zelfs iets totaal anders.

```
PROGRAM:WIEWINT2
:Prompt L1
:max(L1)→M
:Disp "WINNERS:"
:For(I,1,dim(L1))
:If M=L1(I):Then
:Disp I
:End
:End
```

Exercises

- ① Maak **FORDISP** na, maar gebruik 'While' i.p.v. 'For'!
 - Zie je de gelijkenis? Beide statements zorgen voor herhaling.
- ② Zelfde vraag, maar gebruik nu Goto en Lbl.
 - Dat kost veel moeite... Gelukkig bestaan For en While!
- ③ Vind hoe vaak 2 in **Prompt X** past.
 - Bijv.: in het getal 40 past '2' ^{*}3x.
- ④ Vind alle priemgetallen kleiner dan **Prompt X**.
 - Een priemgetal is alleen (integer) deelbaar door 1 en zichzelf.
(2, 3, 5, 7, 11, 13, 17, 19, 23, ...)
- ⑤ Ontbind het getal **Prompt X** in priemgetallen. (Prime Factorisation)
 - Bijv.: $84 = 2^2 \cdot 3 \cdot 7$.
 - Zie ook: <http://www.2dtx.com/prime/prime84.html>
- ⑥ Schrijf het 'risk dobbelstenen' programma van de lecture over pseudocode m.b.v. Lists.

MAAK ALTIJD EERST PSEUDOCODE!

```
PROGRAM: FORDISP
: For(I, 1, 3, 1)
: Disp "FOR"
: End
```

Outline

- 1 Goto & Labels
- 2 Herhaling statements
 - For-loops
 - While-loops
 - Pause
- 3 Datatype: Lists
 - List datatype
 - List hulpjes
 - Matrix datatype
- 4 Exercises
 - Exercises
 - Answers

*

WIEWINT3: Wie worden er tweede en derde?

Een mogelijk antwoord (er zijn zoals altijd meer manieren!)

Algorithm 2 “WieWint3? met Lists”

```
1: function WIEWINT( $L_1$ : list met scores)
2:   Init.  $L_2$  als  $L_1$ : iedereen doet mee voor rank 1
3:   for rank 1, 2, 3 do
4:     Vindt punten voor huidige rank (uit  $L_2$ ): M
5:     Clear  $L_2$  voor hergebruik *
6:     for alle spelers uit  $L_1$  do
7:       if punten = M then Deze speler wint!
8:       else if punten < M then
9:         Deze speler is wellicht een rank lager!
10:        Voeg hem toe aan  $L_2$ 
11:      end if
12:    end for Pause
13:  end for
14: end function
```

```
PROGRAM: WIEWINT3
: Prompt L1
:  $L_1 \rightarrow L_2$ 
: For(J,1,3)
: max( $L_2$ )  $\rightarrow$  M
: {min( $L_1 \rightarrow L_2$ 
: Disp "RANK:", J
: Disp "PEOPLE:"
: For(I,1,dim( $L_1$ ))
: If M= $L_1(I)$ : Then
: Disp I
: Else
: If M> $L_1(I)$ : Then
: augment( $L_2$ , ( $L_1(I)$ 
: ))  $\rightarrow L_2$ 
: End: End: End
: Pause: End
```

(augment = samenvoegen)

Antwoord: FORDISP

- Deze opdracht is een mooi voorbeeld voor het principe “the right tool for the job”.
- Merk op dat “vroeger” mensen geen For en While hadden. De computer begreep alleen Goto. Lucky you! *
- Het gebruik van Goto op onderstaande manier is “bad practice” (don't do it).
 - “Using a Goto to exit any block of code requiring an End command causes a memory leak, which will not be usable until the program finishes running or executes a Return command, and which will slow your program down.”
 - Zie ook: <http://tibasicdev.wikidot.com/goto>

```
PROGRAM:FORDISP
:For(I,1,3,1)
:Disp "FOR"
:End:Pause
:
:1→I
:While I≤3
:Disp "WHILE"
:I+1→I
:End:Pause
:
:1→I
:Lb1 A
:Disp "GOTO"
:I+1→I
:If I≤3:Then
:Goto A
:End
```

Antwoord: Hoe vaak past 2 in x?

Algorithm 3 “Hoe vaak past 2 in X?”

```
1: function HOEVAAK2(X)
2:   Doe wat sanity-checks & initialiseer
3:   while X is een geheel getal ( $\equiv$ integer) do
4:     Vervang X door  $X/2$ .
5:     Houd een tellertje bij, I.
6:   end while
7: end function
```

```
PROGRAM:HOEVAAK2
:Prompt X
:If fPart(X)≠0
or X=0:Then
:Disp 0
:Stop
:End
:
:-1→I
:While fPart(X)=
0
:X/2→X
:I+1→I
:End
:Disp I
```


Antwoord: Vind alle priemgetallen $\leq X$

Algorithm 4 “Vind alle priemgetallen $\leq X$.”

```
1: function PRIMES( $X$ : max. waarde om te checken)
2:   Init. een list,  $L_1$ , om de primes bij te houden
3:   for  $I$ : Alle integers  $\leq X$  do
4:     Neem à priori aan dat  $I$  prime is.
5:     for  $J$ : Alle primes kleiner dan  $I$  do
6:       if  $I$  deelbaar is door  $L_1(J)$  then
7:          $I$  is geen prime. Break loop.
8:       end if
9:     end for
10:    Indien  $I$  niet 'geen prime' is, voeg toe aan  $L_1$ 
11:  end for
12: end function
```

```
PROGRAM: PRIMES
: Prompt X
: ( $2$ ) $\rightarrow L_1$ 
: For( $I$ , $3$ , $X$ )
:  $1\rightarrow P$ 
: For( $J$ , $1$ ,dim( $L_1$ ))
: If fPart( $I/L_1(J)$ 
: )=0: Then
:  $0\rightarrow P$ 
: dim( $L_1$ )+1 $\rightarrow J$ 
: End
: End
: If P: Then
: augment( $L_1$ , $I$ )
:  $\rightarrow L_1$ 
: End: End
: Disp  $L_1$ 
```

Antwoord: Prime Factorisation van x

Algorithm 5 Pseudocode Prime Factorisation

- 1: **function** PRIMEFAC(X : het te factoriseren getal)
- 2: **for** P : Alle primes $< \sqrt{X}$ **OF** Alle natuurlijke getallen $< \sqrt{X}$ **do** \triangleright Beide opties geven hetzelfde resultaat. Het eerste is sneller voor grote X , het tweede voor kleine X . De tweede optie is makkelijker.
- 3: **if** X deelbaar door P **then**
- 4: Store (=Onthoud/Bewaar) P
- 5: Ga verder met X/P
- 6: Herhaal de loop voor P
- 7: **end if**
- 8: **end for**
- 9: **end function**

```

PROGRAM: PRIMEFAC
: Prompt X
: If X=0: Then
: (0)→L1: Disp L1
: Stop: End
: X/abs(X)→S
: XS→Y
:
: (S)→L1
: For(P, 2, int(√(Y)))
: Y/P→Z
: If fPart(Z)=0
: Then
: augment(L1, {P})
: →L1
: Z→Y: P-1→P
: End: End
: Disp L1
    
```

Antwoord: Risk Dobbelstenen I

Algorithm 6 Pseudocode Risk Dobbelstenen

```
1: function RISKDICE( $N_{att}$ )
2:   Maak  $N_{att}$  random getallen tussen 1 en 6
3:   Vraag of  $N_{def}$  1 of 2 is
4:   Maak  $N_{def}$  random getallen tussen 1 en 6
5:   Sorteer beide setten van hoog naar laag
6:   if Getal 1 van set 1 > Getal 1 van set 2 then  $p_1 = p_1 + 1$ 
7:   else  $p_2 = p_2 + 1$ 
8:   end if
9:   if Getal 2 van set 1 > Getal 2 van set 2 then  $p_1 = p_1 + 1$ 
10:  else  $p_2 = p_2 + 1$ 
11:  end if
12:  Display de scores:  $p_1$  en  $p_2$ 
13: end function
```

Antwoord: Risk Dobbelstenen II

```
PROGRAM:RISKDICE
:0→A:0→D
:While A<1 or A>3
:Input "NATT? ",A
:End
:seq(randInt(1,6),X
,1,A,1)→L1
:SortD(L1)
:Disp L1
:
:While D<1 or D>A
or D>2
:Input "NDEF? ",D
:End
:seq(randInt(1,6),X
,1,D,1)→L2
:SortD(L2)
:Disp L2
:
```

```
:0→P:0→Q
:If L1(1)>L2(1)
:Then:P+1→P
:Else:Q+1→Q
:End
:If D>1:Then
:If L1(2)>L2(2)
:Then:P+1→P
:Else:Q+1→Q
:End:End
:
:ClrHome
:Disp L1,L2
:Disp "ATT LOSES:",Q
:Disp "DEF LOSES:",P
```