Unit 4: Lussen | **Toepassing: Berichten van de bank**

Deze toepassing maakt gebruik van lussen om zoveel variabelen als nodig in te voeren en zorgt, als uitbreiding, voor een controle op de geldigheid van variabelen en maakt gebruik van "If"-opdrachten om een passend bericht weer te geven.

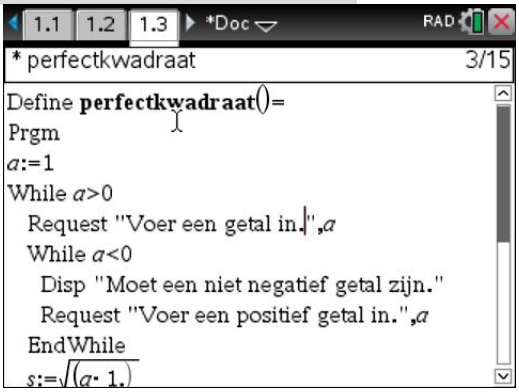
Doelen:

- De opdrachten teller (Counter) en accumulator- gebruiken.
- Een lus gebruiken in een programma om een onbepaalde hoeveelheid gegevens te verkrijgen
- Een 'vlag'-waarde gebruiken om een lus te beëindigen.

Docenten Tip: Dit project illustreert de neiging tot complexiteit bij het ontwikkelen van een stuk software. *Nesten (nesting)* hangt samen met het idee om de ene besturingsstructuur op te nemen binnen een andere. Zoals hieronder wordt uitgelegd weet het OS (besturingssysteem) welke End bij welke opdracht hoort.

Geneste structuren

- **Nesten** is de programmeertechniek waarbij de ene besturingsstructuur binnen een andere wordt geplaatst**.**
- De term is afgeleid van het idee van het plaatsen van de ene kartonnen doos in de andere om zo ruimte te besparen.
- Een programmeur plaatst lussen binnen lussen, **If**-structuren binnen lussen en lussen binnen **If**-structuren om de complexere taken die nodig zijn voor het programma te realiseren.
- Het is belangrijk om een *volledige* structuur *in zijn geheel* binnen een blok van een andere structuur te plaatsen om fouten te voorkomen.
- Het programma rechts toont een **While**-lus en een **If**-structuur binnen een andere **While**-lus.
- Merk het veelvuldig gebruik van de opdracht **EndWhile** op; de computer 'weet' welke **EndWhile** bij welke **While** hoort.
- Het inspringen van de regels is alleen gedaan voor het visuele effect; het helpt om de logica binnen het programma te verduidelijken.



Het programma stelt eerst een lus in om door te gaan totdat de ingevoerde waarde gelijk is aan nul. Vervolgens gaat het na of **a<0**. Als dat zo is toont het programma het bericht "Moet een niet negatief getal zijn!" en vraagt het om een andere waarde voor **a**. Wanneer **a** echter niet negatief is dan worden de wortelberekening, ook een **If**-opdracht, en alle **Disp**-opdrachten uitgevoerd.

Samenvatting van de drie lussen:

For(*var*, *begin*, *eind***)**	**While** <*voorwaarde*>	**Loop**
		If <*voorwaarde*> **: Exit**
EndFor	**EndWhile**	**EndLoop**

For(wordt gebruikt bij 'tellen' of bij het verwerken van een rekenkundige rij van waarden (iteratie).

While wordt gebruikt wanneer je mogelijk de volledige kern van een lus kunt overslaan.

Loop wordt gebruikt wanneer je zeker weet dat je de kern van een lus minstens een keer wilt uitvoeren; deze moet een **Exit**-opdracht bevatten, gewoonlijk als deel van een **If**-opdracht.

Docenten Tip: De enige 'noodzakelijke' lus is de While-lus. Deze kan dezelfde taken uitvieren als de andere twee lussen. Gebruik nooit **GoTo** om een lusstructuur te verlaten en gebruik dit ook niet op een andere plek ergens in het programma. Dat is een voorbeeld van slecht programmeren!
Er bestaan ook andere besturingsstructuren in het menu **Besturing**, die niet worden behandeld in deze serie lessen. Zie de handleiding van TI-Nspire voor meer informatie.

De variabele **a** wordt vermenigvuldigd met 1,0 (in het Engels met 1.) in de wortel-stap om te zorgen dat de uitvoer hetzelfde is op zowel een numeriek als op een CAS platform. Het gebruik van een decimale komma (punt) in een uitdrukking forceert dat het resultaat een benadering is.

Unit 4 Toepassing: Programma "Berichten van de bank"

En klant van een bank heeft verschillende rekeningen bij dezelfde bank. De bank eist een minimaal gemiddeld saldo van €1000 op alle rekeningen om geen servicekosten te hoeven betalen.

Als het gemiddelde tussen €1000 en €1250 ligt dan stuurt de bank de klant een bericht met de waarschuwing dat de klant mogelijk servicekosten moeten gaan betalen.

Wanneer het gemiddelde boven €1250 komt stoort de bank een bericht naar de klant om te bedenken voor het bewaren van een goed gemiddeld saldo.

We gaan een programma schrijven dat de gebruikers informatie geeft over hun rekeningen. De gebruiker voert de saldi van de rekeningen in. Het programma zal dan de gemiddelde saldi bepalen en het aantal rekeningen, de gemiddelde saldi en een passend bericht weergeven. Dit bericht kan zijn:

"SERVICE FEE CHARGED" ("ER WORDEN SERVICEKOSTEN IN REKENING GEBRACHT"),

"IN DANGER OF A SERVICE FEE" (U LOOPT HET RISICO SERVICEKOSTEN TE MOETEN BETALEN),

en "THANK YOU FOR YOUR BUSINESS!" (BEDANKT VOOR UW ZAKEN).

We kunnen twee manieren gebruiken voor het in laten voeren van een onbekend aantal waarden:

- Manier 1: Vraag eerst om het totaalaantal rekeningen en gebruik een **For**-lus om de waarden in te voeren.
- Manier 2: Vraag om de bedragen, maar gebruik een 'vlag'-waarde, bijvoorbeeld -999, om aan te geven dat er niet nog meer rekening zijn. Deze manier maakt gebruik van een **While**-lus of een **Repeat**-lus

In beide gevallen zullen we een lopend totaal van de ingevoerde waarden moeten bijhouden.

Bij manier 2, moeten we ook de saldi *tellen* zodat we het *totaal* door dat *aantal* kunnen delen, de volgende informatie zal nuttig zijn …

Jouw programma zou het volgende moeten weergeven: 1) het aantal saldi dat is ingevoerd, 2) het gemiddelde van de saldi en 3) een passend bericht gebaseerd op het gemiddelde saldo

Als het gemiddelde lager is dan 1000:

"SERVICE FEE CHARGED" ("ER WORDEN SERVICEKOSTEN IN REKENING GEBRACHT")

… 1000 tot 1250:

"DANGER…" ("U LOOPT RISICO…….")

… hoger dan 1250: "THANK YOU…" ("BEDANKT…..")

Docenten Tip: In het gedeelte hieronder bespreken we twee gerelateerde programmeer-ideeën: een teller en een 'accumulator-variabele'. Benadruk dat de variabele links van de toekenningsoperator (:=) dezelfde variabele is als die aan de rechterkant, maar dat zij verschillende waarden voorstellen door de manier waarop de waarden verwerkt worden.

Tellers en accumulatoren

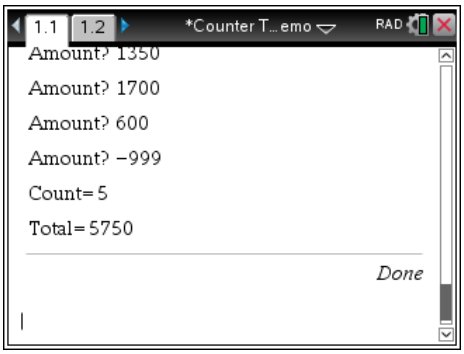
Een opdracht zoals **c:=c+1** wordt een 'teller' genoemd omdat deze, telkens wanner hij wordt uitgevoerd, 1 optelt bij de variabele **c**.

Een opdracht zoals **t:=t+n** wordt een 'accumulator' genoemd omdat deze een lopend totaal van de waarden van de variabele **n** bijhoudt. De waarde van **n** wordt opgeteld bij de variabele **t** en vervolgens wordt die som weer 'terug' opgeslagen in variabele **t**. Aan het eind van een lus zal **t** het totaal van de **n** waarden bevatten

Hieronder zie je een voorbeeld (in het Engels) waarin een teller (counter), een accumulator (total) en een 'vlag'-waarde (-999) worden gebruikt om het aantal bedragen bij te houden dat ingevoerd wordt in een programma:

Prgm	Aantekeningen
Local counter,amount,total	Variabelen initialiseren
total:=0	
counter:=1	
Request "Amount?",amount	Krijg het eerste bedrag (amount)
While amount≠−999	Zolang (while) dit niet -999 is
counter:=counter+1	tel 1 op bij de teller (counter)
total:=total+amount	Tel het bedrag (amount) op bij het totaal (total)
Request "Amount?",amount	Vraag om nog een bedrag (amount)
EndWhile	
Disp "Count=",counter	
Disp "Total=",total	
endPrgm	

Voorbeelduitvoer



De **While**-lus hierboven blijft tellen en de bedragen optellen zolang er geen -999 wordt ingevoerd. Zodra -999 wordt ingevoerd stopt de lus en worden de resultaten weergegeven.

Docenten Tip: Wijs op de twee **Request**-opdrachten in het programma hierboven. De eerste haalt het eerste bedrag op en de laatste vraagt de rest van de bedragen. De laatste staat *onderin* de lus om de terugkeer naar de opdracht **While** voor te bereiden om de waarde van het bedrag opnieuw te laten testen.

Uitbreiding

Als onderdeel van je invoer-routine kun je een controle uitvoeren om er zeker van te zijn dat het bedrag dat is ingevoerd een geldig bedrag is (groter dan 0) en om een passende actie uit te voeren wanneer dat niet zo is.

Teacher Tip: Deze uitbreiding vraagt om een extra lus om elk van de invoeropdrachten geen om er zeker van te zijn dat de ingevoerde waarde geldig is. Het voorbeeldprogramma hieronder stopt simpelweg het programma wanneer er een ongeldige waarde is ingevoerd. Leerlingen kunnen een betere oplossing bedenken!

Docent: Hier zie je een mogelijke oplossing (in het Engels)

```
Prgm
local amount, counter, total, avg
0→ amount
1→counter
0→total
Request "ENTER A balance amount: ", amount
While amount ≠ -999
  If amount <0 Then
    Disp "OUT OF RANGE!"
    Stop
  ElseIf
    counter+1→counter
    total+ amount →total
  End
  Request "ANOTHER amount (OR -999): ", amount
EndWhile
total/counter→avg
Disp "NUMBER OF accounts:", counter
Disp "TOTAL OF balances:", total
Disp "AVERAGE balance:",avg
If avg<1000 Then
  Disp "SERVICE FEE CHARGED")
ElseIf avg<1250 Then
  Disp "IN DANGER OF A SERVICE FEE"
```

Wanneer je een programma test probeer dan juist de extreme gevallen uit zoals het invoeren van -999 als het eerste bedrag en het ergens invoeren van negatieve waarden.

Als het programma dan een foutmelding geeft, heeft de programmeur geen rekening gehouden met alle mogelijke Het programma hierboven zal falen wanneer -999 wordt ingevoerd als eerste waarde, omdat het zal proberen 0 te delen door 0, wat niet kan. De berekening van het gemiddelde zou in een test moeten worden ‘verpakt’ om zeker te weten dat er minstens een bedrag is ingevoerd. Hieronder staat een mogelijk oplossing om dit te realiseren:

```
If count>0
Then
  Avg:=total/counter
ElseIf
  Disp “NO AMOUNTS ENTERED!”
  Stop
End
```

De opdracht **Stop** wordt in dit programma gebruikt om een programma *onmiddellijk* te beëindigen. Deze opdracht kan

```
Else
  Disp "Thank you for your  business!"
EndIf
EndPrgm
```

overal in een programma worden toegevoegd zodat de uitvoering wordt onderbroken en de boodschap ‘Klaar’ verschijnt.