Handouts bij "beginnend programmeren in Lua"

1. Hoofdscherm

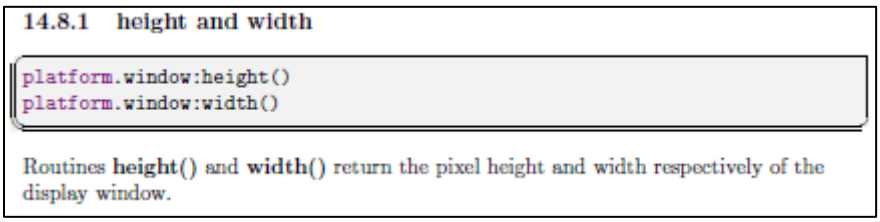
1.1. Inleiding on.paint(gc) functie

In het eerste tabblad worden de grafische elementen die het scherm kunnen opvullen overlopen.Hiervoor moet er gebruik gemaakt worden van een functie: `function on.paint(gc)`

Een functie wordt altijd vooraf gegaan door '`function`', meteen gevolgd door de naam van de functie en tussen haakjes de argumenten die bij deze functie horen. In een functie zit een blok code/reeks instructies die uitgevoerd worden wanneer deze functie aangeroepen wordt. Het einde van een blok code wordt gesloten met '`end`'.

Hier heet de functie '`on.paint`' en is het argument '`gc`'. Die 'gc' staat voor *graphics context* en zal ervoor zorgen dat de gewenste elementen op het scherm verschijnen. In een on.paint(gc)-functie moet álles komen te staan wat op het scherm getekend moet worden.

Van elk element moet de plaats van zowel de x- als y-coördinaat afzonderlijk bepaald worden. Door de hoogte en breedte van het scherm continue te laten opvragen kunnen Strings, figuren, invoervelden, … ook mee vergroten/verkleinen met de schermgrootte. Deze methode is vooropgesteld door Lua.



14.8.1 height and width

```
platform.window:height()
platform.window:width()
```

Routines **height()** and **width()** return the pixel height and width respectively of the display window.

Figuur 1: breedte en hoogte opvragen van scherm

Bij veranderingen aan het scherm wordt veelal een stukje code toegevoegd die het scherm laat herladen naar de nieuwe toestand. Dit kan aan de hand van de instructie '`platform.window:invalidate()`'. Dit stukje code wordt aan het einde van de on.paint-functie net voor '`end`' getypt.

De volgende visuele instructies komen aan bod:

- Tekst (String) op het scherm plaatsen
- Figuren tekenen
- Afbeelding invoegen

1.2. Tekst

Hier wordt er gebruik gemaakt van de instructie 'drawString', wat letterlijk betekent 'teken tekst':

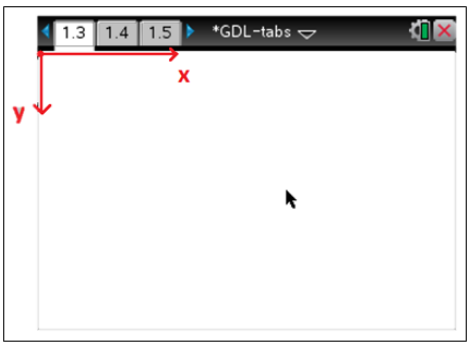
9.7 drawString

```
gc:drawString("text", x, y [,vertalignment])
```

Draws text on the window beginning at pixel location (x,y). Vertical alignment may be "baseline", "bottom", "middle", or "top". This aligns the text in the height of the characters' bounding rectangle. If the vertical alignment is unspecified, it defaults to "none".

Figuur 2: 'drawString' Lua Scripting Reference Guide

De instructie '`drawString`' heeft minstens 3 argumenten nodig: ("tekst",x-coord,y-coord), zoals te zien op figuur 1. Het vierde argument staat tussen vierkante haakjes en is niet verplicht. Het eerste argument is de string die op het scherm moet komen, het tweede en derde argument zijn de coördinaten die bepalen waar de string moet afgebeeld worden op het scherm. Hiertoe ligt er een denkbeeldig assenstelsel op het scherm met oorsprong in de linkerbovenhoek. De x- en y-as liggen zoals aangeduid op figuur 3.



Figuur 3: assenstelsel Ti-nspire scherm

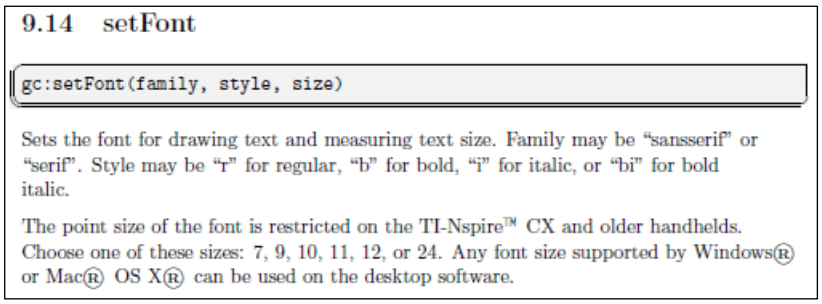
De String kan ook volgens opmaak aangepast worden. De kleur, het lettertype, de lettergrootte en de stijl van de tekst kunnen aangepast worden.

9.13 setColorRGB

```
gc:setColorRGB(red, green, blue)
gc:setColorRGB(0xRRGGBB) - platform.level = '2.0' only
```

Sets the color for subsequent draw and fill routines. The red, green, and blue components of the color are values in the range of 0 to 255. Black is 0,0,0 and white is 255,255,255. Alternately, a single value can be passed in. The components of this single value are blue + 255 * (green + 255 * red).

Figuur 4: kleur aanpassen met 'RGB'-code



9.14 setFont

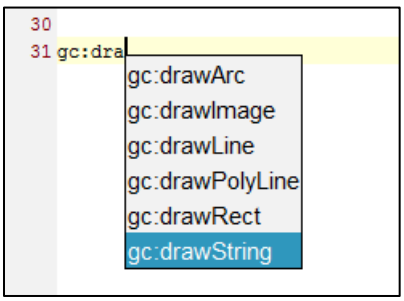
```
gc:setFont(family, style, size)
```

Sets the font for drawing text and measuring text size. Family may be "sansserif" or "serif". Style may be "r" for regular, "b" for bold, "i" for italic, or "bi" for bold italic.

The point size of the font is restricted on the TI-Nspire™ CX and older handhelds. Choose one of these sizes: 7, 9, 10, 11, 12, or 24. Any font size supported by Windows® or Mac® OS X® can be used on the desktop software.

Figuur 5: font aanpassen

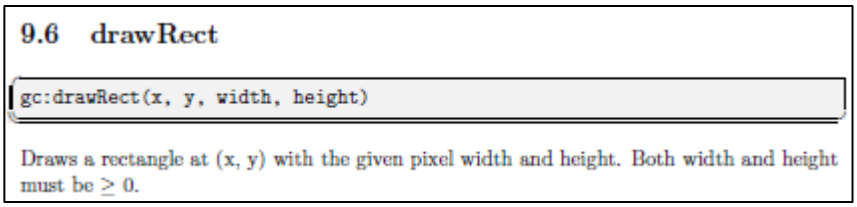
Tijdens het programmeren met Lua kan er gebruik gemaakt worden van 'code completion'. Dit helpt de gebruiker bij het aanmaken van instructies. Door 'Ctrl+spatie' in te drukken in de Lua script editor, opent zich een menu met mogelijke aanvullingen. Met de muis of pijltjes kan een optie gekozen worden. Dit hulpmiddel laat de gebruiker toe om snel en zonder fouten bepaalde instructies te gebruiken. Dit is te zien in figuur 6.



Figuur 6: toepassen 'code completion'

1.3. Figuren

Hiervoor wordt er gekozen welke figuur op het scherm moet komen zoals een rechthoek. Er is wel een verschil tussen een 'draw'-figuur en 'fill'-figuur, de ene tekent enkel de rand van de figuur en de andere tekent een opgevulde figuur met de gekozen RGB-code kleur.

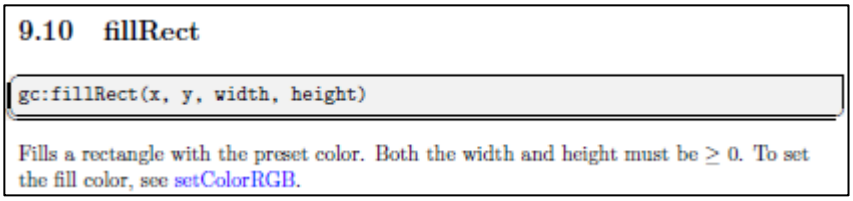


9.6 drawRect

```
gc:drawRect(x, y, width, height)
```

Draws a rectangle at (x, y) with the given pixel width and height. Both width and height must be ≥ 0.

Figuur 7: omranding van een rechthoek



9.10 fillRect

```
gc:fillRect(x, y, width, height)
```

Fills a rectangle with the preset color. Both the width and height must be ≥ 0. To set the fill color, see setColorRGB.

Figuur 8: gevulde rechthoek

1.4. Afbeeldingen

Om een afbeelding in het programma te plaatsen moet deze geïmporteerd worden. Eerst en vooral moet de gekozen afbeelding opgeslagen zijn op de computer, vergelijkbaar met het invoegen van een afbeelding in word.

In de Lua Script editor is er onderaan een paneel met “tools”. Bij het tabblad “hulpmiddelen” kan een afbeelding ingevoegd worden. Als er verder geklikt wordt op “+” dan opent zich een map en daaruit kan dan gezocht worden naar de gewenste afbeelding.

Figuur 9: importeren afbeelding

Daarna moet een de afbeelding in het script toegevoegd worden. De afbeelding wordt opgeslagen in een variabele die zelf te kiezen is. Bijvoorbeeld onze ingevoegde afbeelding “Luaafbeelding” wordt opgeslagen in de variabele “Afbeelding”. De code wordt dan: Afbeelding = image.new(_R.IMG.Luaafbeelding)
Nu moet de afbeelding nog in de on.paint-functie komen zodat het ook effectief op het scherm verschijnt. Hierbij wordt een kopie van de geïmporteerde afbeelding gemaakt en deze laten aanpassen aan de grootte van het scherm door middel van een schaalfactor

```
local h = platform.window:height()          -- definieren hoogte scherm
local w = platform.window:width()           -- definieren breedte scherm
```

Figuur 10: Algemene variabelen on.paint-functie

```
-- 3) geïmporteerde afbeelding op scherm weergeven:
local imw = Afbeelding:width()                  -- opvragen van breedte afbeelding
local imh = Afbeelding:height()                 -- opvragen van hoogte afbeelding
local sf = math.max(imw/(w/2), imh/(h/2))       -- schaalfactor bepalen
local image = Afbeelding:copy(imw/sf, imh/sf)   -- op schaal brengen van figuur
local imw = image:width()
local imh = image:height()
gc:drawImage(image,(w-imw)/2,imh/2)
```

Figuur 11: Afbeelding op schaal brengen en tekenen

2. Menu

2.1. Centrale programma variabele

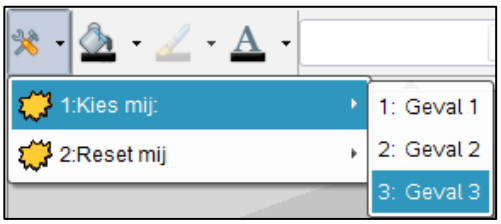
De gebruiker komt bij het openen van het programma op het startscherm en kan daaruit via het menu een gepast hoofdstuk openen. Daarna kan er eventueel in het menu een mogelijkheid zijn om terug te keren naar het hoofdscherm.

Een centrale 'kies'-variabele wordt gebruikt om te verspringen van scherm (bijvoorbeeld verspringen van wiskunde oefening reeksen).

Eerst en vooral is er een centrale variabele nodig welke zijn waarde afhangt van het gekozen hoofdstuk in het menu. Zo wordt er een bepaald "gewicht" meegegeven aan een scherm. Zo kan er makkelijk genavigeerd worden tussen de hoofdstukken (binnen de code). De gebruiker kan (zonder dat hij het weet) de variabele laten variëren door een bepaald hoofdstuk/geval te selecteren in het menu. De centrale variabele past zich aan aan het gekozen geval; bijvoorbeeld bij het aanklikken van geval 1 wordt de centrale variabele ook 1 en dan wordt de code aangeroepen die bij dat geval/hoofdstuk hoort.

In het voorbeeld wordt de centrale variabele '`centraleVariabele`'. In het begin van het programma moet een defaultwaarde meegegeven worden. Meestal hoort de defaultwaarde bij het hoofd-/beginscherm. De meest logische defaultwaarde is 0. '`local centraleVariabele = 0`'

Eerst moet er nagedacht worden hoeveel hoofdstukken er nodig zijn om het aantal puntjes in het menu te kunnen programmeren (In het programma VitruBeam kan er gevonden hoe je in een menu nog eens submenu's kan maken). Daarna kan het aanmaken van een menu beginnen. In het voorbeeld programma zijn er 3 gevallen aangemaakt.



Figuur 12: menu met 3 gevallen

Vervolgens moet er per geval/hoofdstuk een apart zelf gedefinieerde functie worden geprogrammeerd die de centrale variabele laat variëren en een commando stuurt naar het deel van het programma dat uitgevoerd moet worden.

```
-- Zelfgemaakte functies
function geval1SetTrue()                    -- aanroepen geval 1 in menu
    centraleVariabele = 1
end

function geval2SetTrue()                    -- aanroepen geval 2 in menu
    centraleVariabele = 2
end

function geval3SetTrue()                    -- aanroepen geval 3 in menu
    centraleVariabele = 3
end
```

Figuur 13: zelfgemaakte functies voor alle gevallen

Eventueel als de gebruiker moet kunnen terugkeren naar het hoofdscherm, is er een menukeuze nodig die dit aanbiedt. Daarbij wordt de centrale variabele terug op de defaultwaarde gezet en dit door opnieuw een zelf gemaakt functie te programmeren.

```
function terugkerenHoofdscherm()
    centraleVariabele = 0
end
```

Figuur 14: zelfgemaakte functie om terug te keren naar hoofdscherm

2.2. Menu aanmaken

Deze functies worden gebruikt in de effectieve code van het menu. Het aanmaken hiervan moet niet in een functie maar gebeurt als volgt:

```
-- Menu dat verschijnt op hoofdscherm:
menu =
{
    {"Kies mij: ",
            {" Geval 1",function () geval1SetTrue() end},
            {" Geval 2",function () geval2SetTrue() end},
            {" Geval 3",function () geval3SetTrue() end},
    },


    {"Reset mij", {" Hoofdscherm menu",function() terugkerenHoofdscherm() end},

    }
}
toolpalette.register(menu)
```

Figuur 15: menu programmeren

2.3. elseif-statement

In de on.paint-functie moet nu bij het aanroepen van een geval, het juiste geval op het scherm komen. Dit kan door een elseif-statement te programmeren die in de on.paint functie zich afvraagt welke waarde de centrale variabele heeft. zoals gewoonlijk wordt in de on.paint-functie eerst de hoogte en breedte van het scherm opgevraagd:

```
local h = platform.window:height()          -- definieren hoogte scherm
local w = platform.window:width()           -- definieren breedte scherm
```

Figuur 16: hoogte en breedte scherm opvragen

```
if centraleVariabele == 0 then

    local string1 = "2. HOOFDSCHERM MENU"
    gc:drawString( string1 , w/8 , h/10 )

elseif centraleVariabele == 1 then

    local string1 = "Dit is geval 1."
    gc:drawString( string1 , w/2.8 , h/2.2 )

elseif centraleVariabele == 2 then

    local string1 = "Dit is geval 2."
    gc:drawString( string1 , w/2.8 , h/2.2 )

elseif centraleVariabele == 3 then

    local string1 = "Dit is geval 3."
    gc:drawString( string1 , w/2.8 , h/2.2 )

end -- deze end beïndigt de if-statements
```

Figuur 17: elseif-statement van de verschillende gevallen

Deze if statements vragen zich telkens de waarde van de centrale variabele af. ‘`centraleVariabele == 0`’ betekent: ‘is de waarde van centraleVariabele gelijk aan 0?’. Zo niet wordt de code overgeslagen die in dit deeltje staat en zal het programma overschakelen naar het volgende if statement: ‘`centraleVariabele == 1`’ en zo verder. Komt de afgevraagde waarde overeen met de werkelijke waarde van de variabele dan wordt het deeltje dat in het if statement staat uitgevoerd. In deze gevallen zal er een String (een stuk tekst) op het scherm verschijnen. Als de waarde van 1 van de if-statements 1 maal overeenkomt met de werkelijke waarde dan wordt een elseif-statement afgesloten. (bvb centraleVariabele = 1; dan wordt over alle code tot ‘`centraleVariabele == 1 then`’ gesprongen. vanaf dan wordt de code in deze elseif-statement uitgevoerd tot het volgende elseif-statement. in plaats van de andere elseif-statements daarna nog te overlopen zal er direct naar ‘`end`’ gegaan worden. Dit wil zeggen dat er maximaal 1 blokje code vanuit de elseif-statements aangeroepen zal worden per keer dat de on.paint functie wordt aangeroepen.

3. Invoervelden

3.1. Aanmaken invoerveld

In een programma is het handig als de gebruiker dingen kan invoeren zoals waarden van in een oefening zoals in VitruBeam. Hier wordt het aanmaken van zo'n invoervelden besproken.

Eerst en vooral moet een invoerveld aangemaakt worden.

```
-- invoervelden aanmaken
Invoerveld1 = D2Editor.newRichText()          --invoerveld aanmaken
Invoerveld2 = D2Editor.newRichText()          --invoerveld aanmaken
```

Figuur 18: invoervelden aanmaken

Vervolgens moeten de invoervelden op het scherm komen. Dit gebeurt opnieuw in de on.paint(gc)-functie. De lay-out van zo'n invoerveld kan zelf worden bepaald zoals grootte (resize), plaatsing op scherm (move), lettergrootte (fontsize), rand of geen rand (setborder) en of de "focus" (setfocus) dus de cursor in dat invoerveld moet staan. Om de invoervelden dan zichtbaar te maken moet de instructie '`setVisible()`' op '`true`' worden geset.

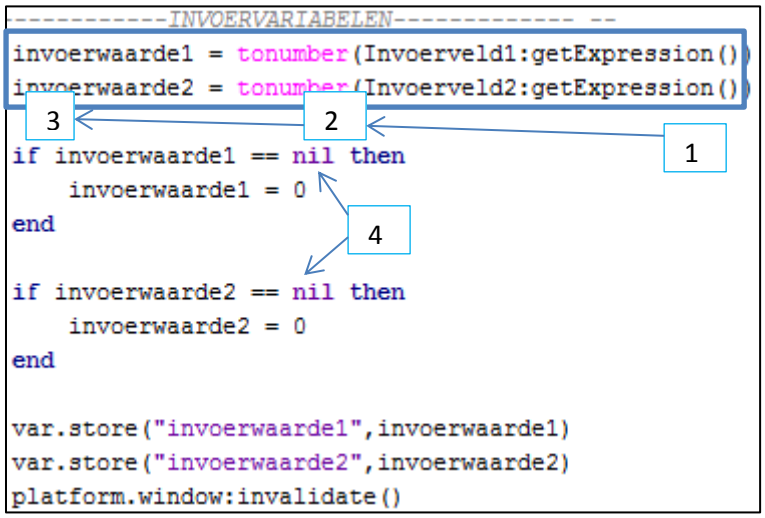
```
Invoerveld1:move(w/5, h/2.5)                -- invoerveld 1 positioneren
Invoerveld1:resize(w*0.17, h*0.12)          -- invoerveld 1: grootte bepalen
Invoerveld1:setFontSize(h/22)               -- invoerveld 1: tekstgrootte bepalen
Invoerveld1:setBorder(1)                    -- invoerveld 1: (1) voor rand; (0) zonder rand

Invoerveld2:move(w/1.8, h/2.5)              -- invoerveld 2 positioneren
Invoerveld2:resize(w*0.17, h*0.12)          -- invoerveld 2: grootte bepalen
Invoerveld2:setFontSize(h/22)               -- invoerveld 2: tekstgrootte bepalen
Invoerveld2:setBorder(1)                    -- invoerveld 2: (1) voor rand; (0) zonder rand

-- invoervelden op scherm krijgen
Invoerveld1:setVisible(true)                -- Invoerveld false/true = onzichtbaar/zichtbaar
Invoerveld2:setVisible(true)                -- Invoerveld false/true = onzichtbaar/zichtbaar
```

Figuur 19: aangemaakte invoervelden op scherm plaatsen en zichtbaar maken

3.2. Opslaan van invoer

Na het aanmaken van invoervelden zou het handig kunnen zijn om de gegevens die ingevuld worden op te kunnen slaan in variabelen zodat die gegevens meteen of later gebruikt kunnen worden om bijvoorbeeld een bewerking uit te voeren.

De instructie '`getExpression()`' haalt de gegevens uit het invoerveld op als "tekst", stel dat er met de invoer gerekend moet worden dan moet deze omgezet worden naar getallen "numbers". Dit kan door de instructie '`tonumber()`' te gebruiken met als argument '`getExpression()`'. De waarden van de invoervelden worden opgehaald [1], omgezet naar getallen [2] en dan vervolgens opgeslagen in variabelen [3].

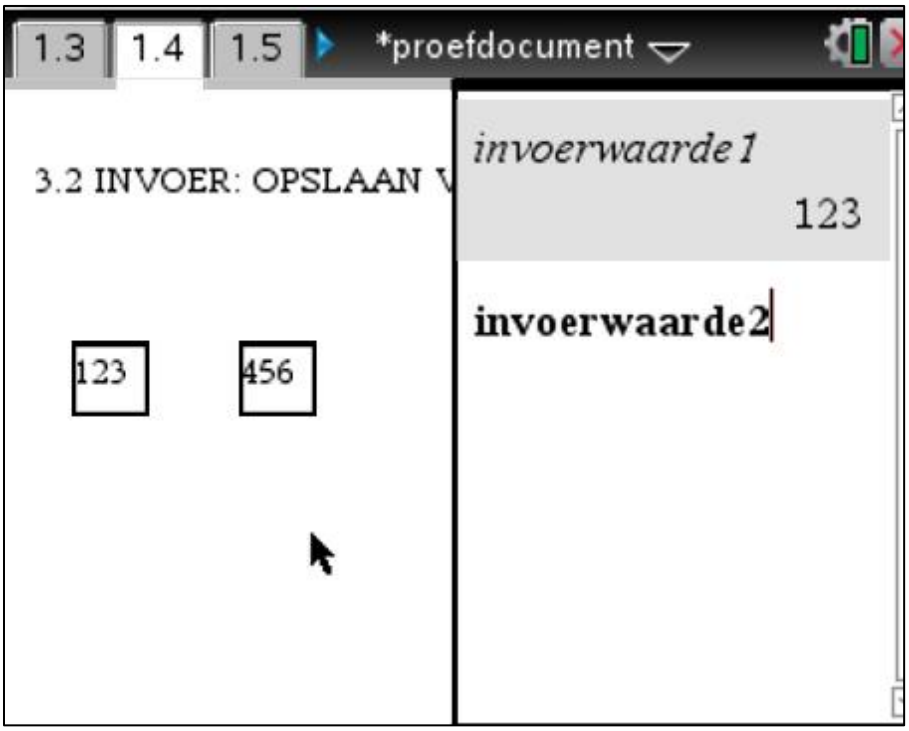


Figuur 20: ophalen en opslaan invoervelden

Indien de gebruiker niks in het invoerveld typt dan wordt er een '`nil`' value opgehaald wat zowel geen getal als String is. Aangezien niet met nil-values gerekend kan worden moet het programma een beetje geholpen worden. We verwachten immers dat als er niks getypt wordt de waarde '0' wordt meegegeven, dit doen we handmatig door het if-statement [4].

Als tijdens het programmeren of uitvoeren van het programma de waarden van de variabelen willen worden verkregen om te kijken of alles werkt, kan dit eenvoudig door ze te "storen" in een Ti-nspire variabele. De instructie var.store werkt als volgt: '`var.store("Ti-nspire variabele",gelinkte Lua variabele)`'. Als in dit voorbeeld de Ti-nspire variabele "invoerwaarde1" wordt en de gelinkte Lua variabele hieraan is '`invoerwaarde1`' (dit kan hetzelfde genoemd worden, maar dit is geen vereiste), dan wordt het stukje code: '`var.store("invoerwaarde1",invoerwaarde1)`'.

Als er dan in een rekenblad *"invoerwaarde1"* wordt getypt dan zal de waarde van de Lua variabele '`invoerwaarde1`' op het scherm verschijnen.

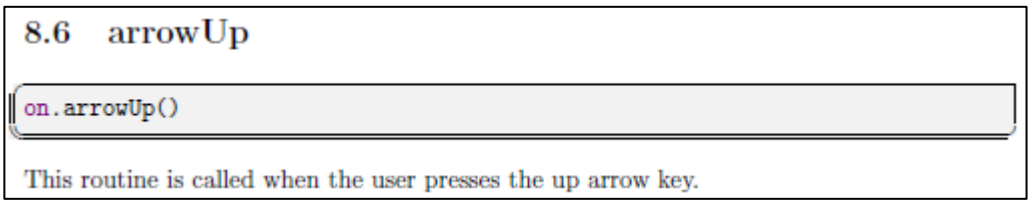


Figuur 21: links Lua programma, rechts rekenmachine tabblad Ti-nspire

4. Event handling – toetsfuncties

4.1. Gebeurtenis

Om bepaalde acties uit te voeren is het handig om bepaalde instructies mee te geven met toetsen zoals pijltjes, escape en tabkey. Deze instructies zijn volledig zelf te bepalen, bijvoorbeeld met de pijltjes toets omhoog (↑) kan er besloten worden om het programma te laten rekenen of om de invoer vanuit de invoervelden op te laten halen. Zo kan het programma onbelast werken tot de gebruiker zelf beslist om het programma te laten rekenen wat de performance alleen maar ten goede komt.



Figuur 22: toetsfunctie 'arrowUp'

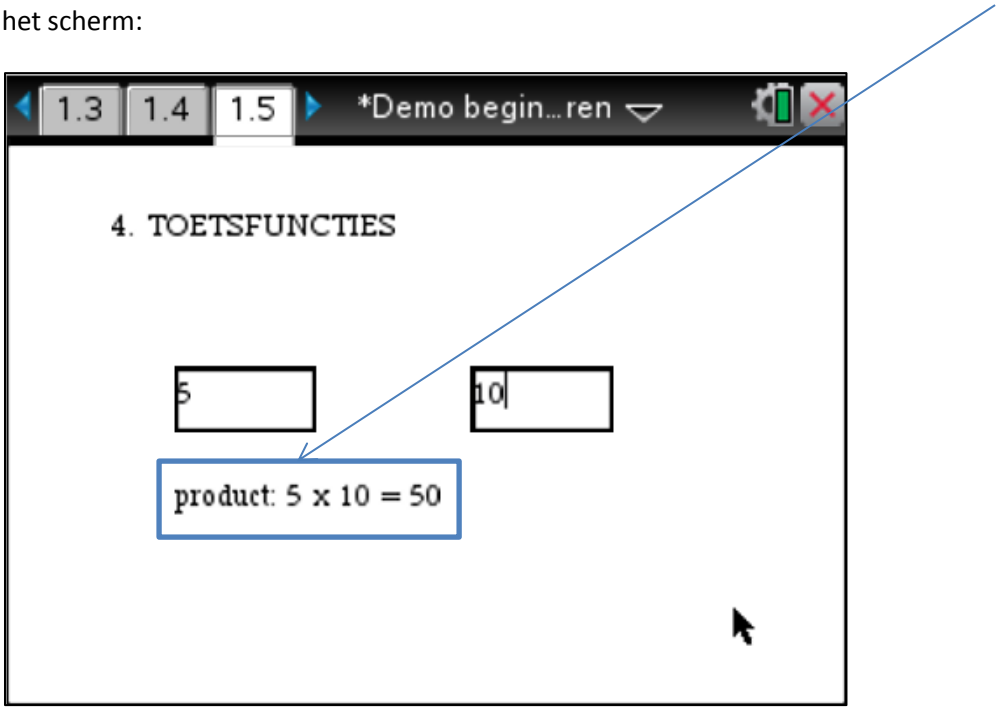
4.2. arrowUp toets om te rekenen

Na het drukken op de pijltjes toets omhoog moet de invoer vanuit de invoervelden opgehaald worden en omgezet worden naar getallen en opgeslagen worden in variabelen zoals gezien in puntje 3.2.

```
function on.arrowUp()
---------------INVOERVARIABELEN------------ --
    invoerwaarde1 = tonumber(Invoerveld1:getExpression())
    invoerwaarde2 = tonumber(Invoerveld2:getExpression())

    if invoerwaarde1 == nil then
        invoerwaarde1 = 0
    end

    if invoerwaarde2 == nil then
        invoerwaarde2 = 0
    end

    var.store("invoerwaarde1",invoerwaarde1)
    var.store("invoerwaarde2",invoerwaarde2)
    platform.window:invalidate()

end
```

Figuur 23: arrowUp methode

Daarna kan er met deze variabelen gerekend worden en het resultaat moet dan "getekend" worden in de on.paint-functie aan de hand van een drawString. In het demoprogramma worden de 2 invoerwaarden met elkaar vermeneigvuldigd. Om tekst en variabelen samen te voegen om een volledige String te maken, kan dit door middel van concatenatie. Tussen de tekst en variabele moeten er dan 2 puntjes '..' getypt worden: '`gc:drawString("product:"..invoerwaarde1.."x"..invoerwaarde2.."="..invoerwaarde1*invoerwaarde2,w/5 , h/1.7)`'. Stel dat invoerwaarde1 = 5 en invoerwaarde2 = 10 dan verschijnt het volgende resultaat op het scherm:



Figuur 24: resultaat van product en concatenatie van String + variabelen

4.3. escapeKey

In het demoprogramma werkt de escape toets als de leegmaaktoets van de invoervelden, zo worden de invoerwaarden gereset.

```
function on.escapeKey() -- maakt elk invoerveld leeg en focust op het eerste invoerveld

        Invoerveld1:setExpression("")            -- elke invoerbox wordt leeg gemaakt
        Invoerveld2:setExpression("")            -- elke invoerbox wordt leeg gemaakt
        platform.window:invalidate()             -- refreshen van scherm

end
```

Figuur 25: escapeKey methode

5. Contact

Bij vragen over Lua of programmeerproblemen mag er altijd contact opgenomen worden via een mail aan Cedric ’t Jampens.

cedric.tjampens@student.kuleuven.be