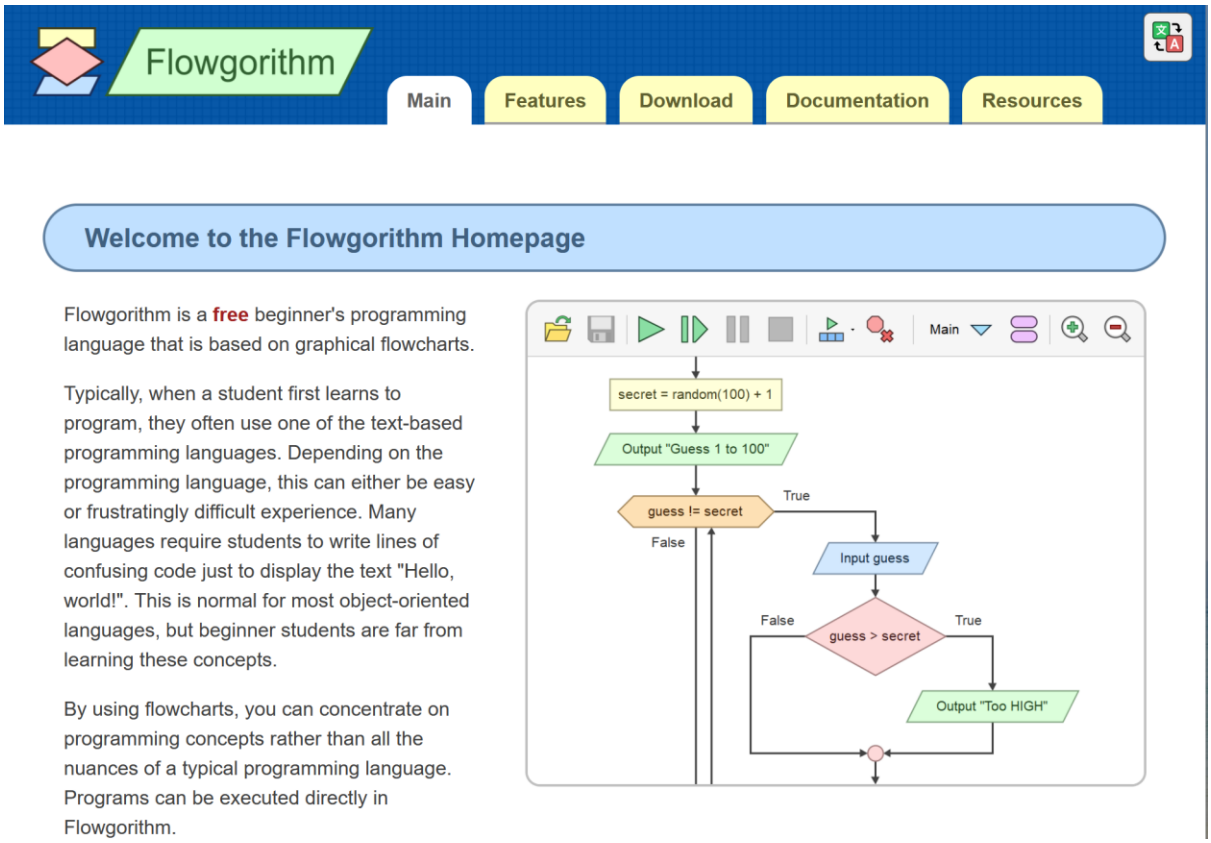
Flowgorithm

Inleiding

Flowgorithm is een *gratis* programmeertool voor beginners die is gebaseerd op eenvoudige grafische stroomdiagrammen.

De webpagina van Flowgorithm is: www.flowgorithm.org



Klik op de tab Download om naar de download pagina te gaan waar je Flowgorithm kunt downloaden om te installeren op je (*Windows*) computer. Kies de laatste versie bij het kopje “Windows Installer”

Basis programmeren met Flogorithm

Wanneer een je voor het eerst leert programmeren, gebruikt je meestal een van de op tekst gebaseerde programmeertalen. Afhankelijk van de taal kan dit eenvoudig of frustrerend moeilijk zijn. In veel talen moet je regels met verwarrende code schrijven om alleen de tekst "Hallo wereld!" weer te geven op het scherm.

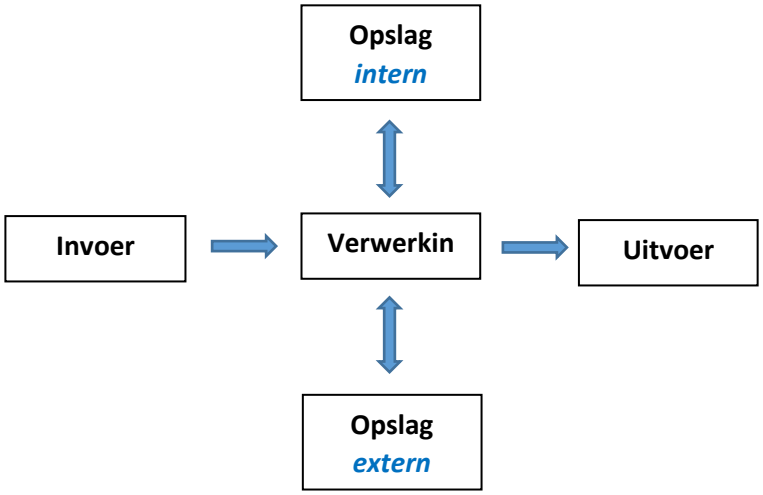
Door stroomdiagrammen (*Flowcharts*) te gebruiken, kunt je concentreren op programmeerconcepten in plaats van alle nuances van een typische programmeertaal. Je kunt je programma's ook rechtstreeks in Flowgorithm uitvoeren.

Flowgorithm

Als je eenmaal programmeerlogica begrijpt, kunt je gemakkelijk een van de belangrijkste talen leren.

Met Flowgorithm kan je stroomdiagram interactief converteren (omzetten) naar meer dan 18 programmeertalen. Bijvoorbeeld C#, C++, PHP, Java, JavaScript, Lua, Perl, Python, Ruby, Swift, Visual Basic .NET en VBA (gebruikt in Office), etc.

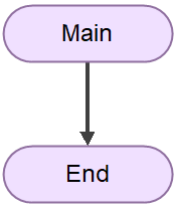
Alle programma's werken volgens het proces dat in het schema hieronder wordt weergegeven.



Programma Proces schema

Zo ook Flowgorithm. Deze heeft symbolen voor de invoer, opslag (*intern*), verschillende verwerkingen, voor de uitvoer en voor de opslag (*extern*).

Elk Flowgorithm (flowchart/stroomdiagram) start als volgt:

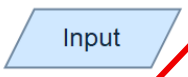


In plaats van "Main" zie je in stroomdiagrammen soms ook wel "Start" staan.

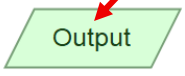
Tussen **Main** en **End** worden de andere elementen van een programma geplaatst.

Stel je wilt de tekst "Hallo wereld!" door je programma op het scherm laten zien. Dit is *uitvoer* en hiervoor is er een symbool in Flogorithm.

Input / Output



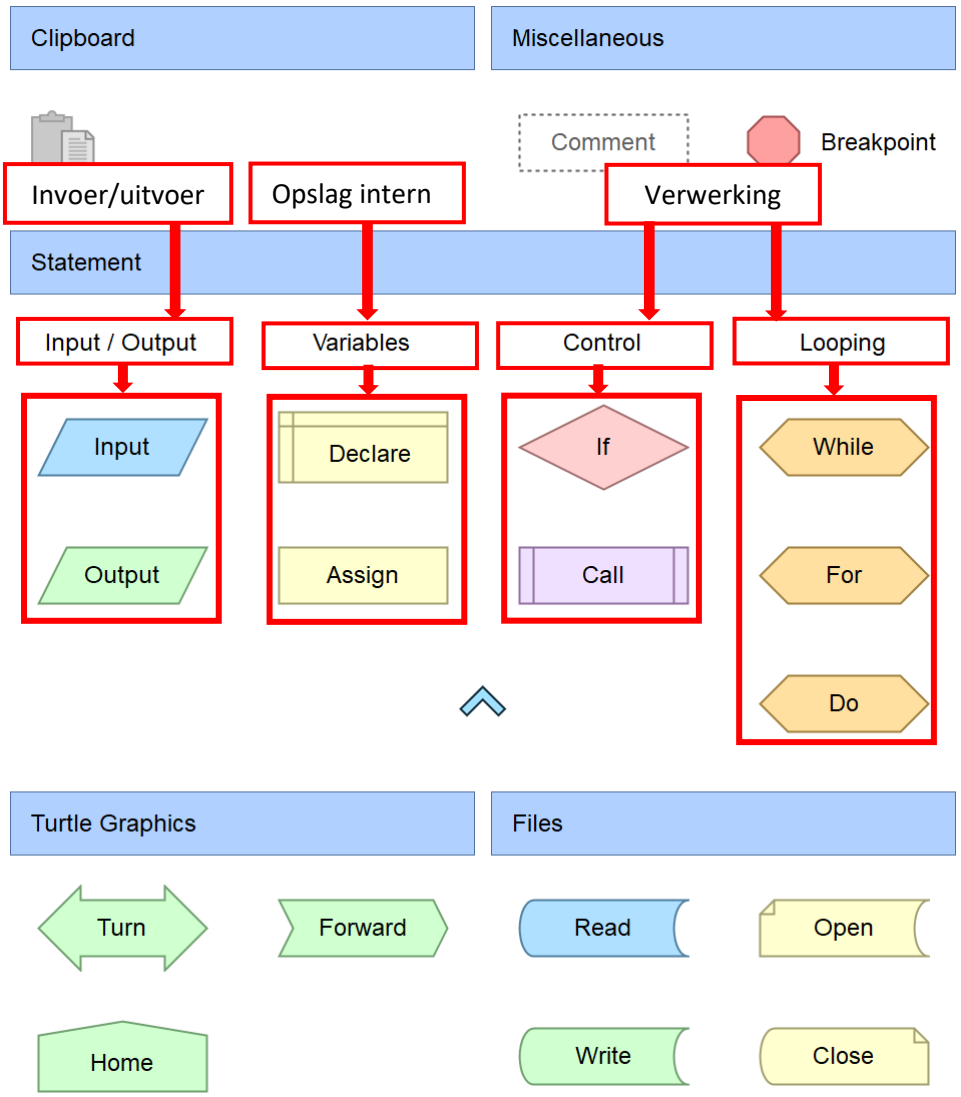
Het input symbool laat de gebruiker van het programma een waarde (*tekst* of *getal*) in het voeren zodat het programma dit kan gebruiken.



Het output symbool zorgt ervoor dat het programma tekst op het scherm kan tonen (*laten zien*) aan de gebruiker.

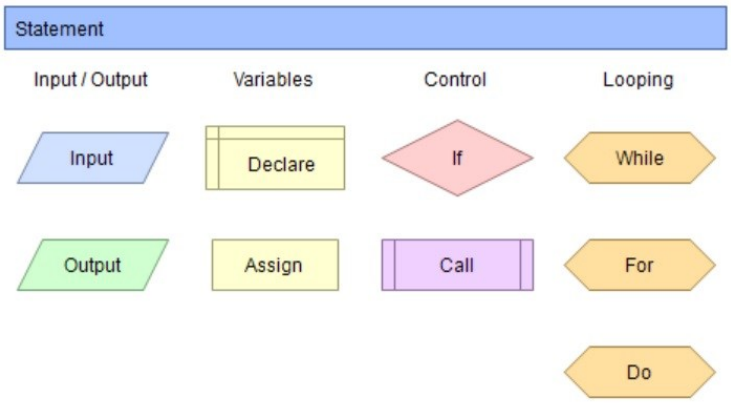
Flowgorithm

Klik op de rechtermuisknop met de muiswijzer op de lijn tussen **Main** en **End**. Je krijgt dan het onderstaande venster te zien:

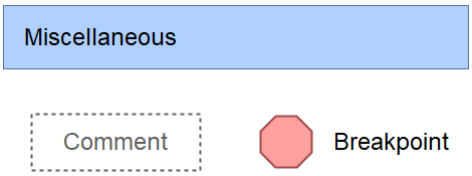


In de afbeelding hierboven ziet men alle verschillende symbolen van Flowgorithm (*versie 4.5*).

In de afbeelding hier naast staan de basis symbolen die gebruikt gaan worden voor het maken van een eenvoudige programma's (*algoritmen*).

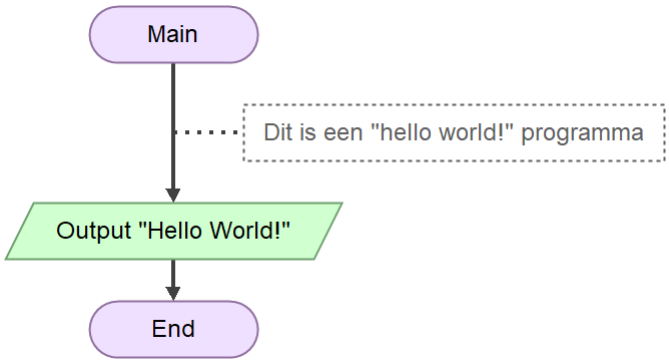


Flowgorithm

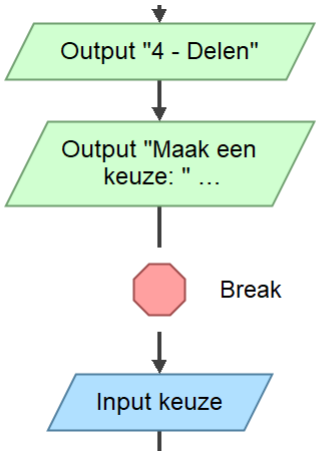


Het **Comment** symbool wordt gebruikt om commentaar een je stroomdiagram toe te voegen.

Het toevoegen van commentaar heeft geen invloed op de werking van het programma maar maakt het wel duidelijker leesbaar voor een ander als je met meer mensen werkt.



Het Breakpoint symbool stopt de uitvoer van het programma op de plek waar je deze gezet hebt. Dit is handig voor het opsporen van fouten in het stroomdiagram, in het Engels zegt men "**Debugging**".

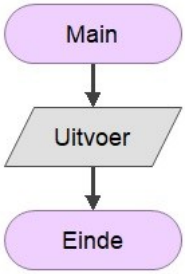


Het Programma wordt gepauzeerd zodat je kunt analyseren wat er tot zover gebeurd is.

Met het step ikoon kun je dan stap voor stap de opdrachten in het stroomdiagram uitvoeren en controleren wat er gebeurt. Dit helpt om eventuele fouten op te sporen en verbeteren.

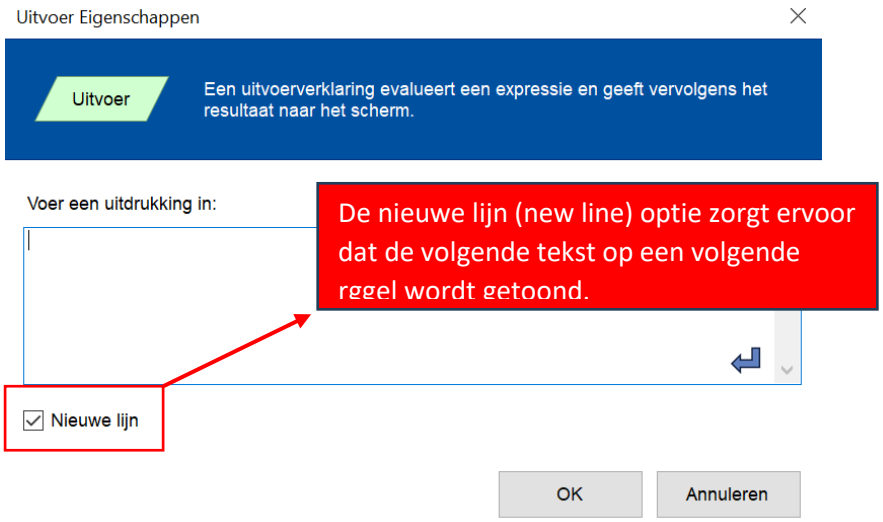
Flowgorithm

Opdracht 1:

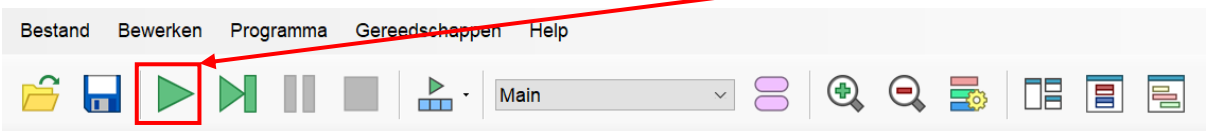


- Plaats een Uitvoer symbool tussen Main en End.

- Dubbelklik op het uitvoersymbool. Je krijgt dan het onderstaande venster te zien:



- Tik in het "voer een uitdrukking in:" vak "Hallo Wereld!" in.
- Klik op de OK knop.
- Voer het programma uit door te klikken op de knop uitvoeren.



- Bekijk het resultaat.

Op deze manier wordt uitvoer aan de gebruiker van het programma getoond.

Variabelen

Om een programma iets zinnigs te laten doen heb je variabelen nodig. Dit zijn locaties in het geheugen waar je informatie kunt opslaan in het interne geheugen om te gebruiken binnen je programma.

Alle verschillende variabelen moeten een unieke naam hebben. Deze mogen bestaan uit letters en cijfer. Ze mogen geen spaties of vreemde tekens bevatten. Volgens de **camelCase** methode beginnen met een kleine letter en elk nieuw woord in de variabele begint weer met een hoofdletter.

Flowgorithm

Denk eraan dat de meeste programmeertalen hoofdletter gevoelig zijn. Dit wil zeggen dat bijvoorbeeld de variabele Naam niet hetzelfde is als de variabele naam.

Kies een naam voor een variabele die aangeeft wat er in de variabele wordt opgeslagen en maak de namen zo kort mogelijk maar wel duidelijk.

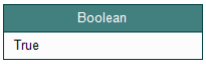
Er zijn verschillende soorten variabelen mogelijk. In Flowgorithm worden deze type variabelen onderscheiden:

- **Integer**, dit zijn gehele getallen bijvoorbeeld -2,-1,0,1,2 etc.

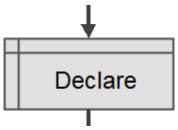
- **Real** (Echt), dit zijn de reële getallen bijvoorbeeld 5.5, -5.5 etc.

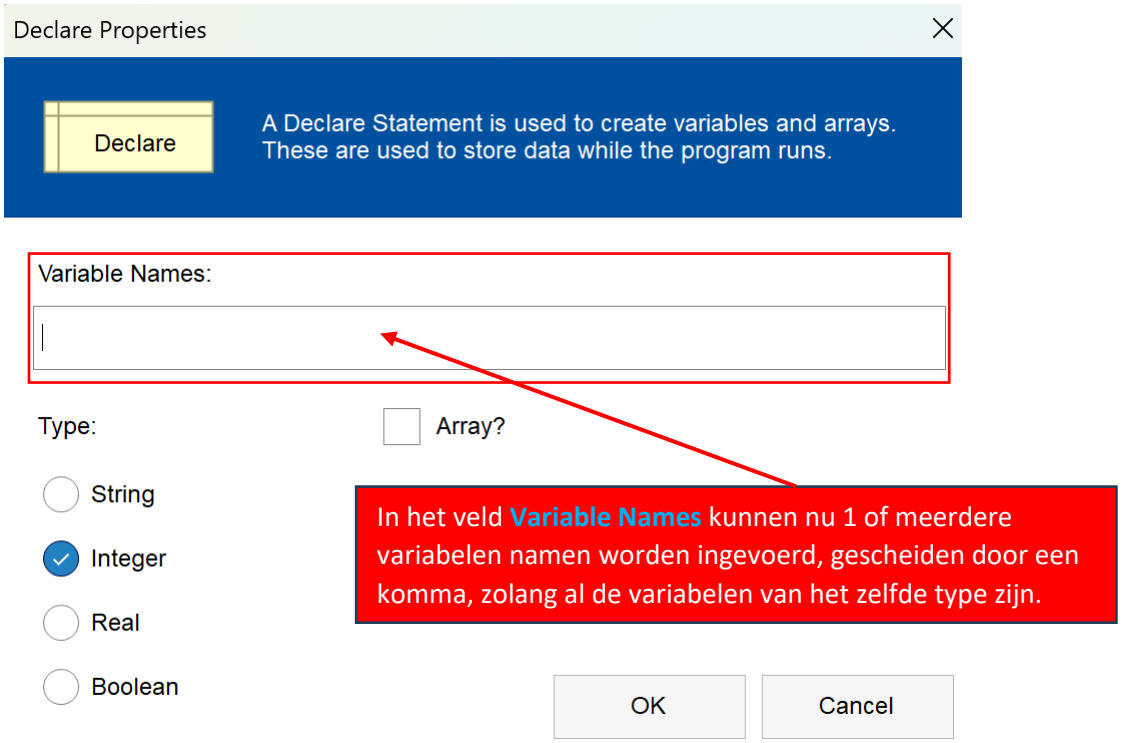
- **String**, dit is tekst (letters en cijfers) tussen dubbele aanhalingstekens. Bijvoorbeeld “Hallo wereld!”.

- **Boolean**, dit zijn variabelen die waar (true) of niet waar (false) zijn.



Wanneer je een Declare blok toevoegt aan je diagram zit het er zo uit. Dubbel klik op dit blok zodat je het dialoogvenster krijgt om je variabel(en) te declareren.





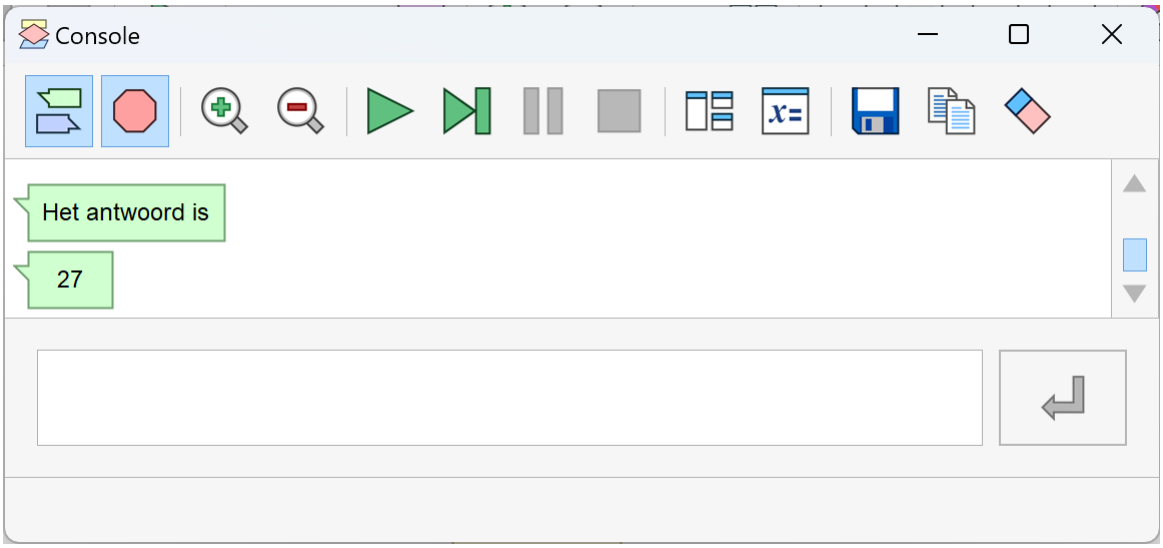
Flowgorithm

Om variabelen te gebruiken in Flowgorithm moet je deze eerst **declareren** (*verklaren*). Zie dit als het plakken van een **label met naam** op een lade van een laden kast.

Daarna Kan een waarde aan de variabele worden toegekend, **assign**, of een berekening worden uitgevoerd waarvan het resultaat aan een variabele wordt toegekend.

Opdracht 2:

- Start een nieuw Flowgorithm programma.
- Voeg de variabelen antw, getalA en getalB van het type integer toe aan het programma.
- Ken (assign) de volgende waarden toe aan de variabelen:
  getalA = 10 en getalB = 17
- Ken de volgende waarde toe aan de variabele:
  Antw = getalA + getalB
- Zorg ervoor dat het programma de berekening uitvoert op de volgende manier:



- Voer het programma uit en controleer of het werkt zoals de afbeelding hierboven.

Opdracht 3:

Bereidt het programma van opdracht 2 uit dat het ook de antwoorden geeft om de twee getallen van elkaar af te trekken, te vermenigvuldigen en te delen. Voer het programma uit en kijk of de resultaten kloppen.

Opdracht 4:

Wat gebeurt er al je voor het getal A bijvoorbeeld 10 invult en voor het getal B 0 (nul)? Wat is het antwoord en leg uit waarom dat.

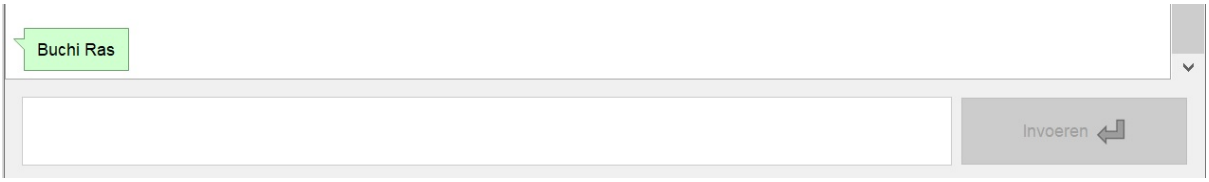
Concatenatie

Soms is het nodig om een regel tekst "string" op te bouwen uit meerdere stukjes tekst. Dit noemen we **concatenatie**. Bijvoorbeeld "Piet " en "Hein" om de tekst "Piet Hein" te kunnen vormen. In Flowgorithm wordt hiervoor **&**-symbool gebruikt. Andere programmeertalen zie je soms **+**-symbool of **;**-symbool hiervoor.

Opdracht 5:

Maak een programma in Flowgorithm waar bij je twee string variabelen declareert, voornaam en achternaam, en deze de waarden toekent van je eigen voor en achternaam. Daarna in één uitvoerblok, de twee namen achter elkaar afdrukt met een spatie tussen beide.

Stel dat voor voornaam gekozen is "Buchi" en voor achternaam "Ras" dan is de uitvoer als volgt:

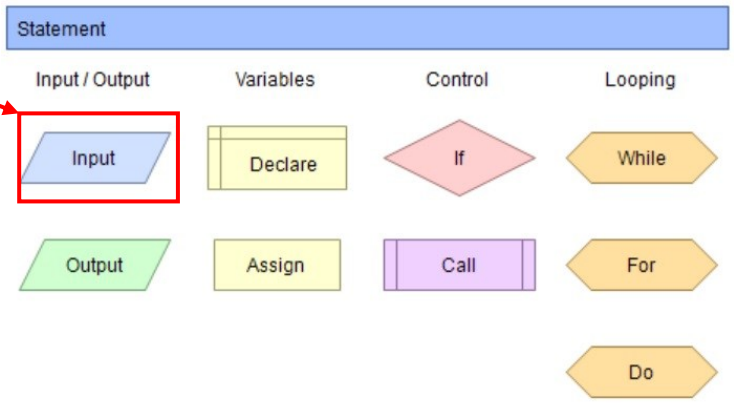


Flowgorithm

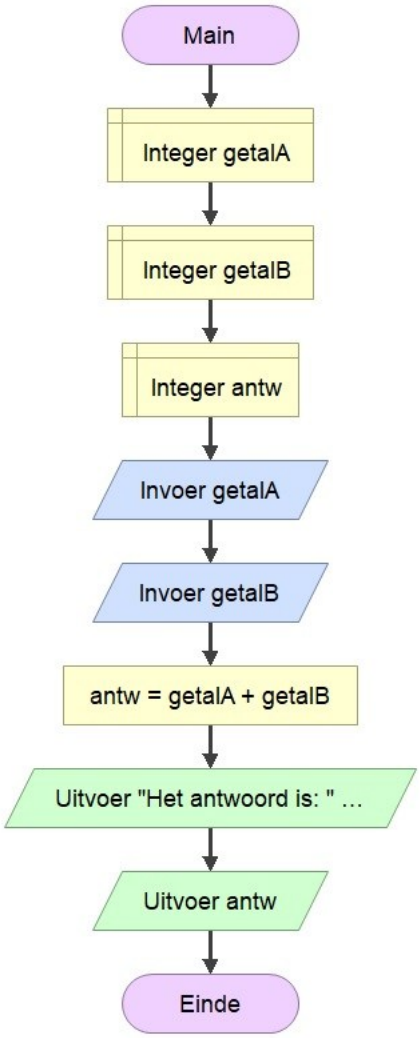
Invoer

Tot nu toe moet je iedere keer als je andere waarden wilt gebruiken het programma aanpassen. Echter het is beter als je de gebruiker van het programma zelf waarden laat invoeren.

Hiervoor gebruik je het invoer/input symbool.



Stel je wilt een programma maken dat de gebruiker twee getalen laat invoeren waarna het deze optelt en het resultaat van de berekening aan de gebruiker toont. Het programma zou er dan als volgt uit kunnen zien:

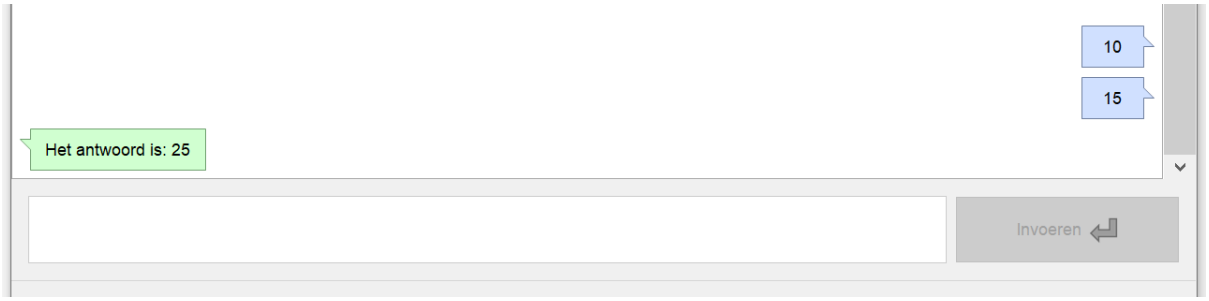


Opdracht 6:

Maak dit programma in Flowgorithm en bewaar het op je desktop met de naam **rekenen.fprg**. De programma's van Flowgorithm hebben als extensie **.fprg**. De extensie moet je niet intypen, dat doet het programma zelf.

Voer het programma uit en bekijk het resultaat goed.

Het resultaat zou bijvoorbeeld kunnen zijn:



Wat viel op?

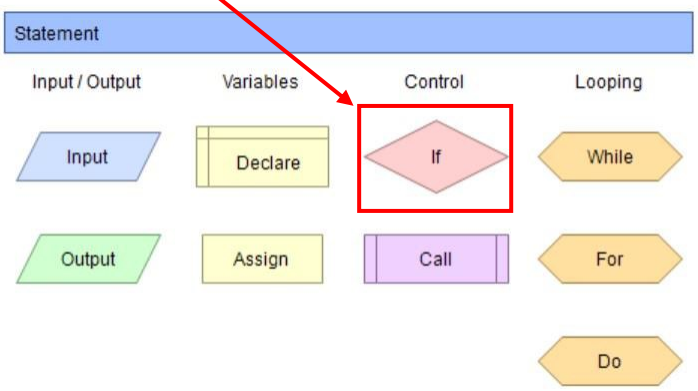
Bij het starten van het programma leek het of er niets gebeurde. Dit is echter niet correct. Het programma was aan het wachten op de gebruiker om een waarde voor het getalA in te voeren. Dit is echter niet gebruikersvriendelijk.

Opdracht 7:

Pas het programma zodanig aan dat het voor de gebruiker duidelijk wordt dat er iets van hem of haar verwacht wordt. Bijvoorbeeld door de tekst "Voer een waarde in voor het getal A: " te tonen voor dat het programma vraagt voor de invoer hiervan.

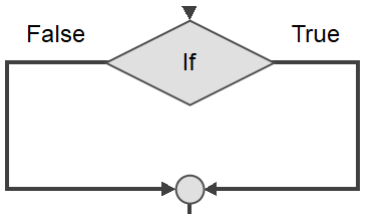
Structuren

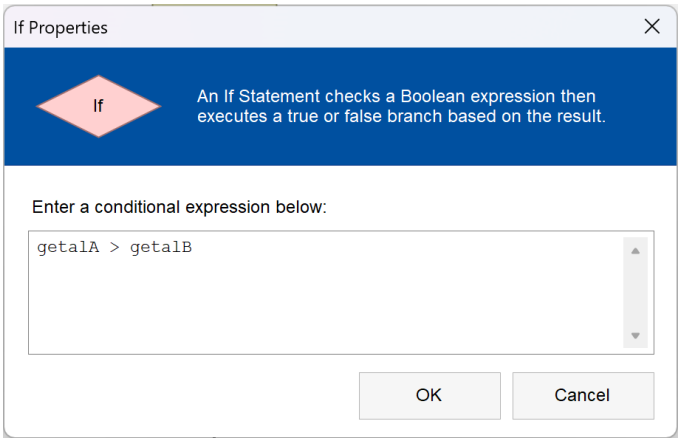
De manier van programmeren die tot nu toe gebruikt is wordt **Sequentie** genoemd. Dit omdat de computer de instructies in volgorde van de eerste naar de laatste uitvoert.
In het geval van flowcharts van **Start/Main** naar **Stop/End**.
Echter zal het heel vaak voorkomen dat je het programma wilt laten kiezen welke instructies het moet uitvoeren. Hiervoor wordt een structuur gebruikt die **Selectie** wordt genoemd. Dit is één van de controle-structuren die de stroom, ***flow***, van een programma beïnvloed.



Flowgorithm

Deze instructie heet de **if**-instructie. Het wordt gebruikt om e bepalen met welke instructies de computer verder moet gaan. Dit gebeurt doormiddel van één of meerdere condities. Als de conditie(s) waar is/zijn dan gaat het programma verder met de uitvoer van instructies aan de rechter kant (**True**). Is dit niet het geval dan worden de instructies aan de linker kant (**False**) uitgevoerd.





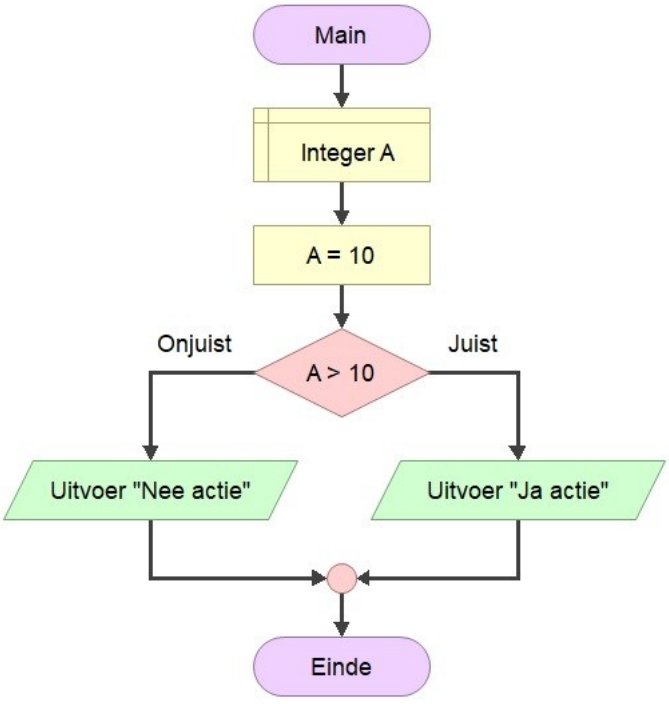
Wanneer je het if-symbool toevoegt en er op dubbel klikt krijg je het dialoogvenster hiernaast te zien.

In dit venster wordt/worden de condities toegevoegd. Zo'n conditie wordt ook wel een **voorwaarde** of de **criteria** genoemd waaraan voldaan dient te worden.

Criteria of wel voorwaarden

Bij een voorwaarde wordt een **logische expressie** vergeleken. Er wordt gekeken of dat deze expressie waar of niet waar is.

De expressie A > 10 wordt gecontroleerd als de waarde die de variabele A heeft groter is dan 10, dan worden de <Ja Opdrachten> uitgevoerd.



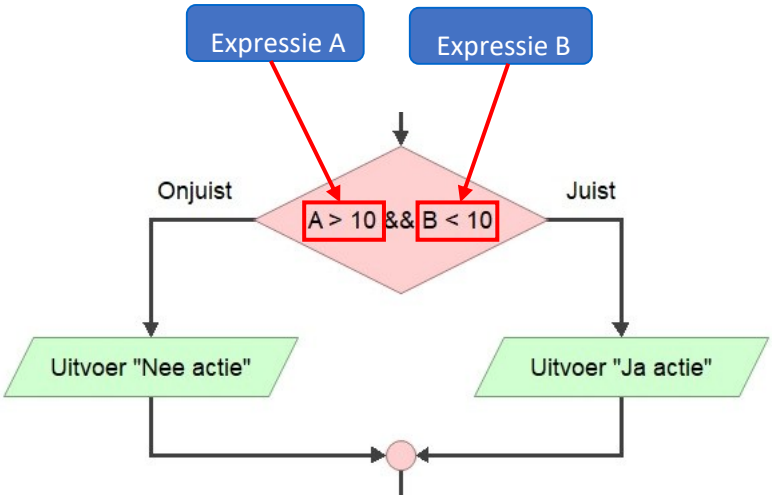
Mogelijke vergelijkingen:

A = B, *gelijk aan*	A < B, *kleiner dan*
A <> B of A != B, *ongelijk aan*	A >= B, *groter of gelijk aan*
A > B, *groter dan*	A <= B, *kleiner of gelijk aan*

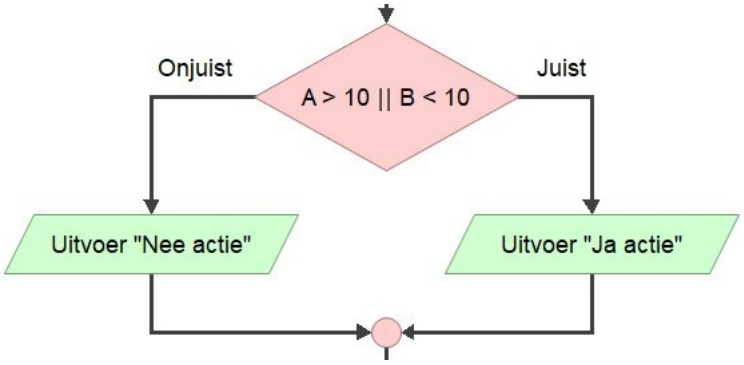
Flowgorithm

Soms moeten er meerdere expressies vergeleken worden, daarvoor zijn er de symbolen/instructies **&&** (AND) **en** || (OR).

<Expressie A> && <Expressie B>



of <Expressie A> || <Expressie B>



De volgende tabellen laten zien wanneer de <Ja Opdrachten> of de <Nee Opdrachten> worden uitgevoerd.

	Expressie A		
Expressie B	AND	W	N
	W	W	N
	N	N	N

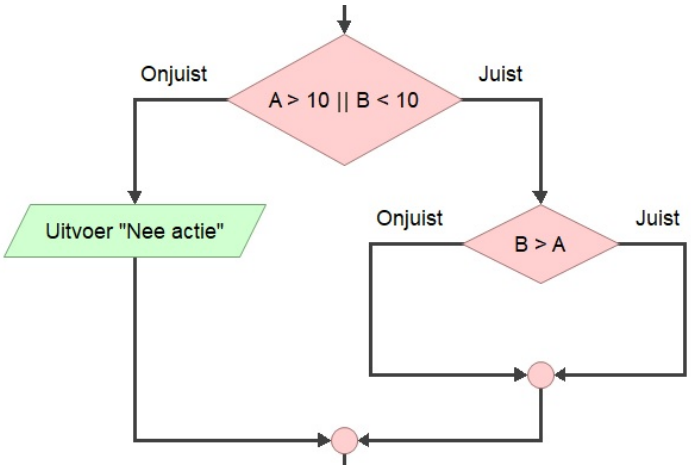
Flowgorithm

	Expressie A		
Expressie B	OR	W	N
	W	W	W
	N	W	N

Bij de AND-instructie moeten beide expressies *waar* zijn wil het resultaat van de totale expressie waar te laten zijn. Bij de OR-instructie moeten beide expressies *niet waar* zijn om het resultaat niet waar te laten zijn.

Nesten

Structuren kunnen genest worden, dit wil zeggen in elkaar gezet worden. Je kunt bijvoorbeeld meerdere selectie structuren in elkaar hebben. Ook iteratie structuren kunnen genest worden. Of iteraties binnen selecties of omgekeerd zijn mogelijk.

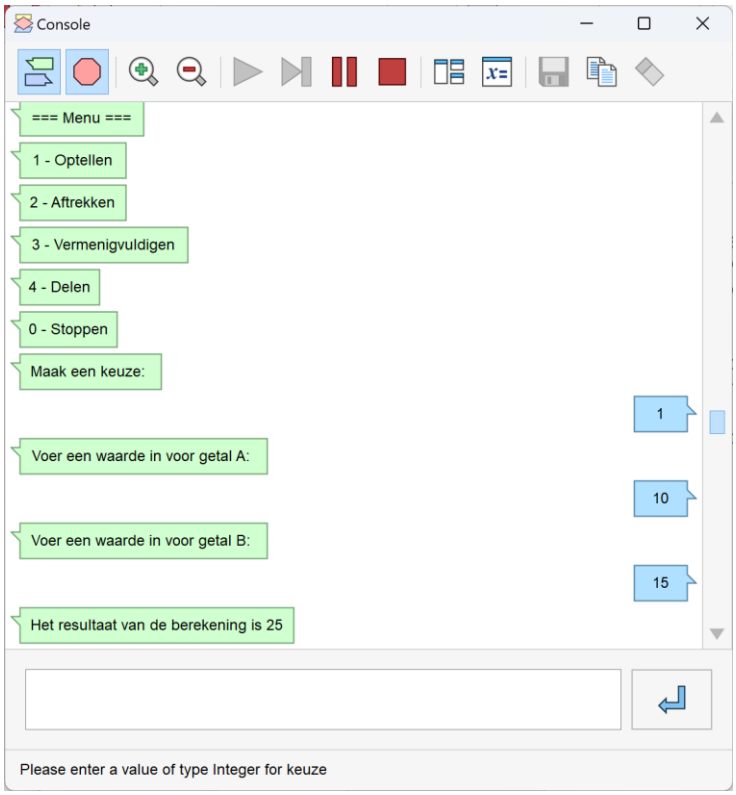


Opdracht 8:

Pas het programma zodanig aan dat het de gebruiker een keuze aanbiedt om of op te tellen of af te trekken, te vermenigvuldigen en te delen. Waarna het programma het resultaat van de berekening toont aan de gebruiker.

Tip: Er is nog een variabele nodig om de keuze van de gebruiker te registreren en te bepalen welke code verder uitgevoerd moet worden.

Het resultaat van het programma zou bijvoorbeeld kunnen zijn:

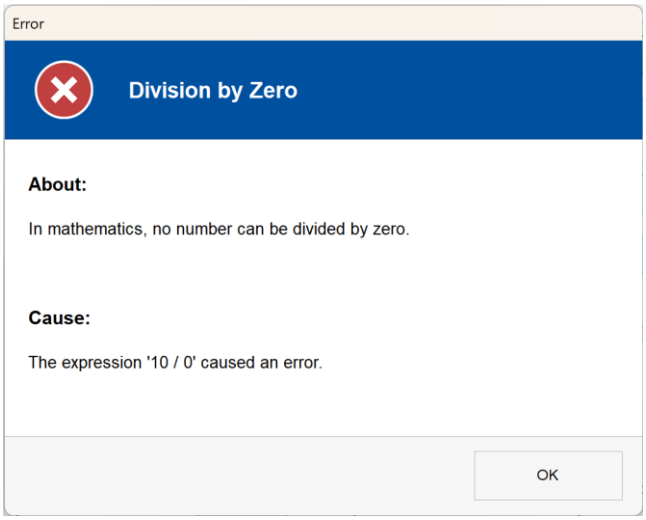


Flowgorithm

De structuur die bij opdracht 8 gebruikt moet worden wordt meestal de **if … then … else** structuur genoemd.

Opdracht 9:

Wanneer je in het programma van opdracht 8 voor het getal A bijvoorbeeld de waarde 10 invult en voor getal B de waarde 0. Zal het programma vast lopen.

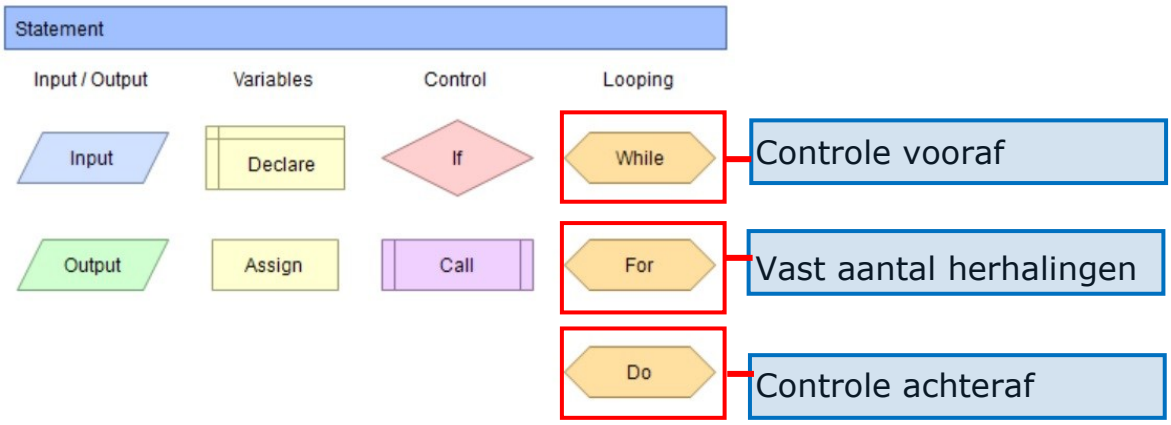


Met behulp van de selectie-structuur is het mogelijk om het probleem bij het delen door nul op te lossen (*zie opdracht 4*). Pas het programma zodanig aan dat dit probleem op een adequate (goede) manier wordt opgelost.

Zorg ervoor dat het programma niet gaat delen door nul maar dan een foutmelding aan de gebruiker geeft.

Itereren of wel herhalen

Het is echter nog beter als de gebruiker het programma niet iedere keer opnieuw hoeft op te starten. Dit is mogelijk doormiddel van een herhaal structuur, de **iteratie**. Ook wel lussen of loops genoemd.
Er zijn drie soorten iteratie structuren, te weten:



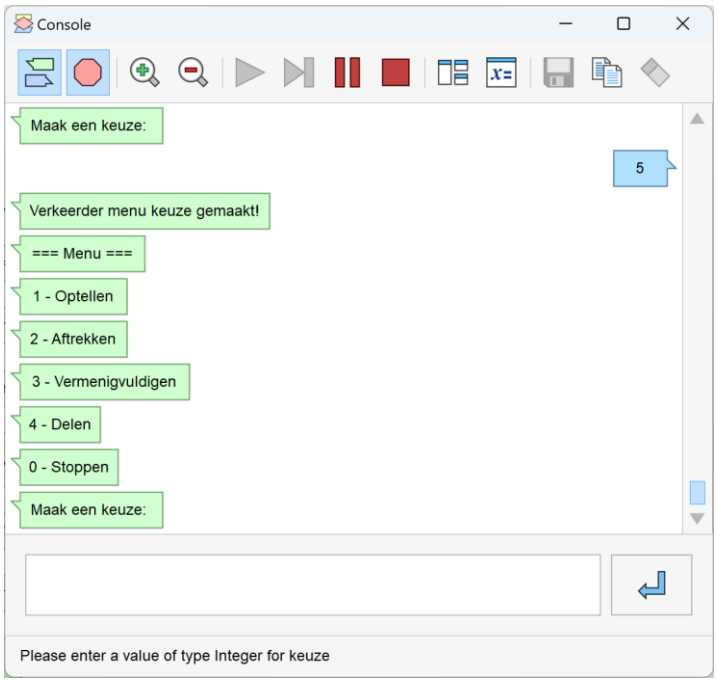
Van de drie iteratie structuren worden in principe alleen de eerste twee gebruikt, **while** en **for**. De derde kan voor fouten/problemen in een programma zorgen daar bij deze de eerste keer de instructies in de lus worden uitgevoerd en pas daarna wordt gecontroleerd of het wel kan.

De while-instructie wordt voornamelijk gebruikt wanneer niet bekend is weet hoeveel keer het programma de lus moet herhalen.
De for-instructie wordt gebruikt wanneer bekend is hoeveel keer de lus moet herhaalt dient te worden.

Opdracht 10:

Pas het programma van opdracht 9 zodanig aan dat het de gebruiker een keuze aanbiedt om of op te tellen of af te trekken of te vermenigvuldigen of te delen en eventueel te stoppen met het berekenen. Het programma dient nu ook aan te geven als er een verkeerde keuze gemaakt is en dient te stoppen als de gebruiker 0 kiest.Sla het programma op en voer het programma uit en bekijk het resultaat.

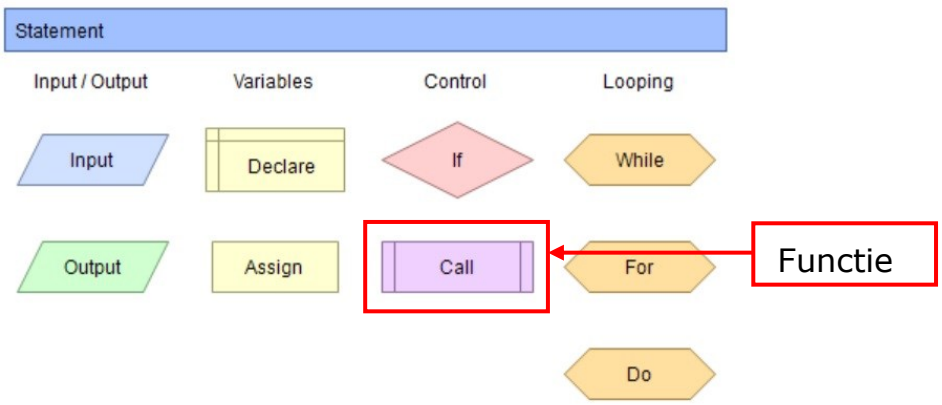
Het resultaat zou er bijvoorbeeld zo uit kunnen zien:



Functies (sub-routines)

Het stroomdiagram voor dit programma is al aardig lang geworden en hoe langer en groter het stroomdiagram wordt hoe minder goed leesbaar deze wordt. Dit geldt straks ook voor programma's waar de code is geschreven en niet doormiddel van symbolen wordt gemaakt.

Om langere programma's toch goed leesbaar te houden wordt van een andere controle structuur gebruik gemaakt. De **functie**, vaak ook subroutine, procedure, module, methode of taak genoemd.



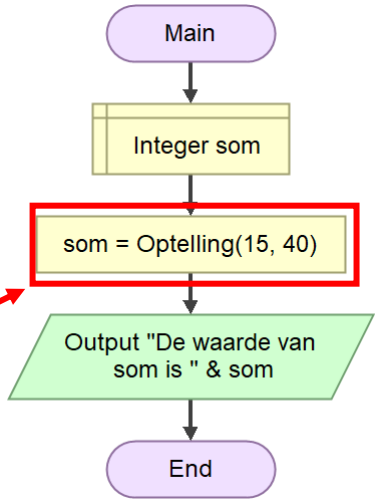
Flowgorithm

Een **functie** is een stuk code dat door een naam wordt aangeroepen. Bij sommige programmeertalen kun je waarden meegeven tijdens de aanroep. Deze waarden worden dan afhankelijk van de soort aanroep in aparte variabelen opgeslagen die **parameters** worden genoemd.

Ook is het mogelijk dat bij sommige programmeertalen zo'n functie een waarde teruggeeft aan het object dat het heeft aangeroepen. Dit is de **return value** (waarde).

Een functie kan op twee manieren gebruikt worden:

1. Het **Call** blok
   - Deze manier is alleen handig wanneer een functie niets terug hoeft te sturen naar de plaats waar het is aan geroepen. Bijvoorbeeld als het alleen een tekst op het scherm moet tonen.
2. In een **Assign** blok
   - Dit is de manier die het meeste gebruikt zal worden. Meestal wil men dat een functie een waarde terug stuurt.



Opdracht 11:

Pas het programma van opdracht 10 zodanig aan dat de werking hetzelfde blijft maar nu gebruik maakt van functies voor het optellen, aftrekken, vermenigvuldigen en delen.

Analyse
Bedenk welke functies nodig zijn, geef ze een goede naam, geef aan wat de eventuele parameters zijn en de eventuele teruggeef waarde ***(return value)***.

Denk er bij het delen aan dat er een melding gegeven moet worden als het getalB gelijk aan nul is. Delen door nul is namelijk niet toegestaan.

Tip: *Namen van functies beginnen met een hoofdletter en elk volgend woord in de naam begint ook weer met een hoofdletter. Dis heet de Camel Case notatie. Kies een niet al te lange naam die weergeeft wat de functie doet.*

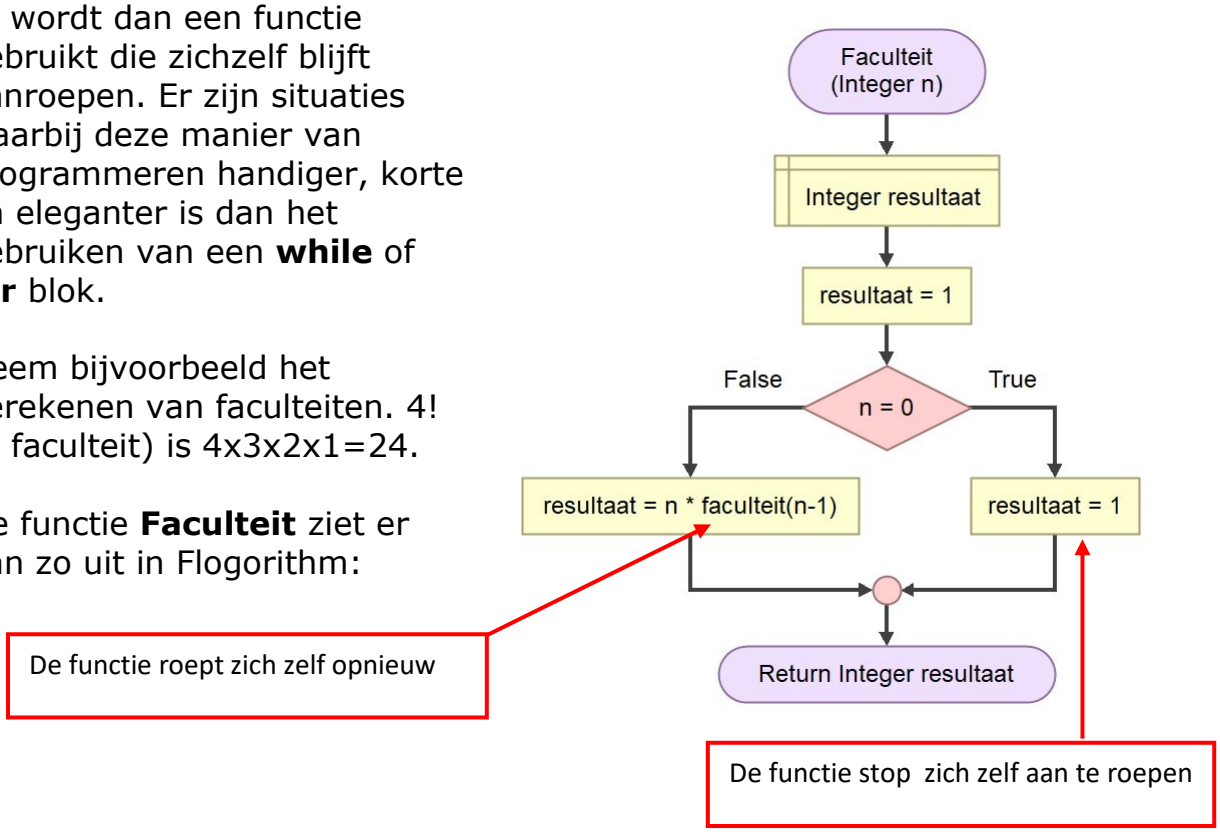
Recursie

Eerder is uitgelegd dat er drie manieren zijn om een iteratie te maken, **while**, **for** en **do**. Echter door gebruik te maken van de call blok is het ook mogelijk om een soort loop te creëren.

Er wordt dan een functie gebruikt die zichzelf blijft aanroepen. Er zijn situaties waarbij deze manier van programmeren handiger, korte en eleganter is dan het gebruiken van een **while** of **for** blok.

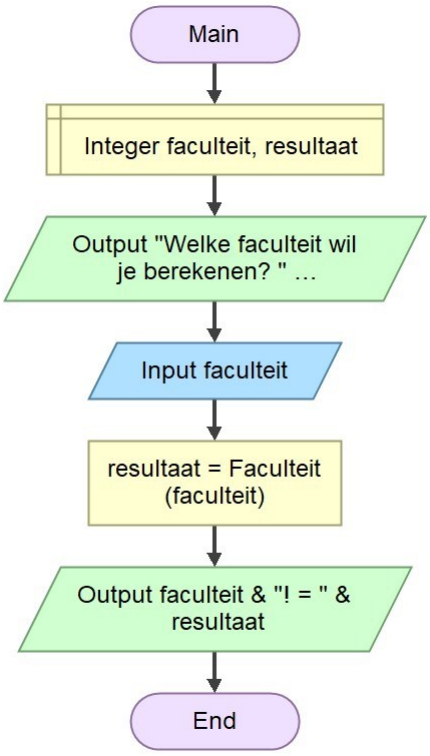
Neem bijvoorbeeld het berekenen van faculteiten. 4! (4 faculteit) is 4x3x2x1=24.

De functie **Faculteit** ziet er dan zo uit in Flogorithm:



Om deze functie uit te proberen is de code in **Main** als volgt:

Flowgorithm



De gebruiker voer hier een faculteit in die berekent moet worden.

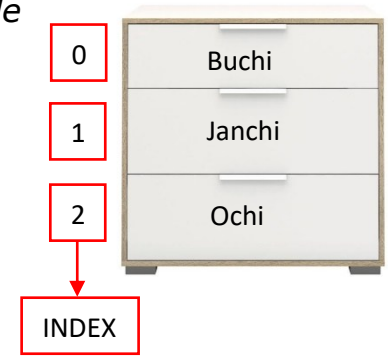
De functie voor het berekenen van de faculteit wordt hier aan geroepen en het resultaat opgeslagen in de variabele resultaat.

Het resultaat wordt op het scherm getoond

Lijsten

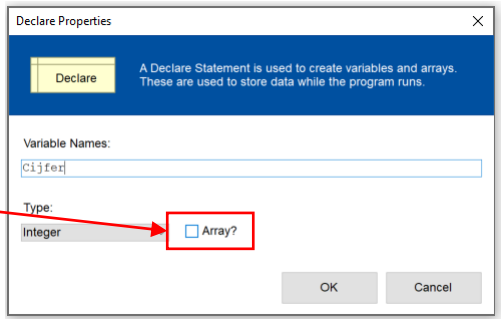
Eén variabele kan één enkele waarde bevatten. Soms is het echter handig om een lijst van dezelfde type met één variabele naam te kunnen gebruiken in een programma. Dit kan doormiddel van een **array**.

Een array is een lijst van variabelen "*van hetzelfde type*" met één naam. Denk aan een variabele als één enkele lade in een ladenkast. Elke lade in de kast kan een enkele waarde bevatten en kan worden aangeroepen door het nummer van de lade, de **index**.



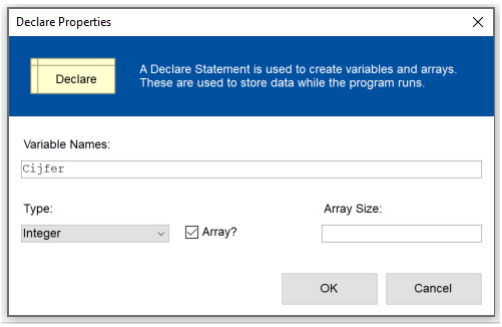
De meeste programmeertalen beginnen de index te tellen bij het cijfer **nul**. Dit wordt **zero based** genoemd.

In Flowgorithm kun je tijdens het declareren van een variabelen aangeven of het een enkele variabele of een lijst variabele moet zijn door naast het type het hokje Array aan te vinken.

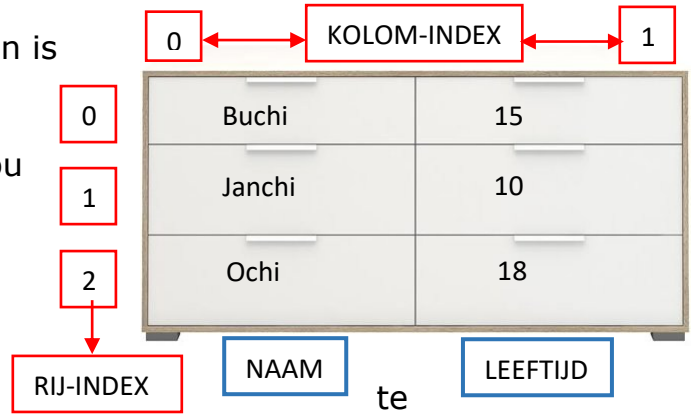


Flowgorithm

Wanneer dat aangevinkt wordt in het declaratie venster, wordt er gevraagd naar de grote van de array. Dit wil zeggen het aantal elementen *(laden in een ladekast)* dat de array moet bevatten.

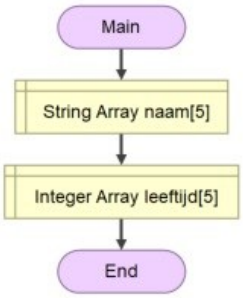


Bij de meeste programmeertalen is het mogelijk om twee of meer dimensionale arrays te maken. Een twee dimensionale array zou dan twee indexen hebben. Eén voor de rijen en één voor de kolommen.

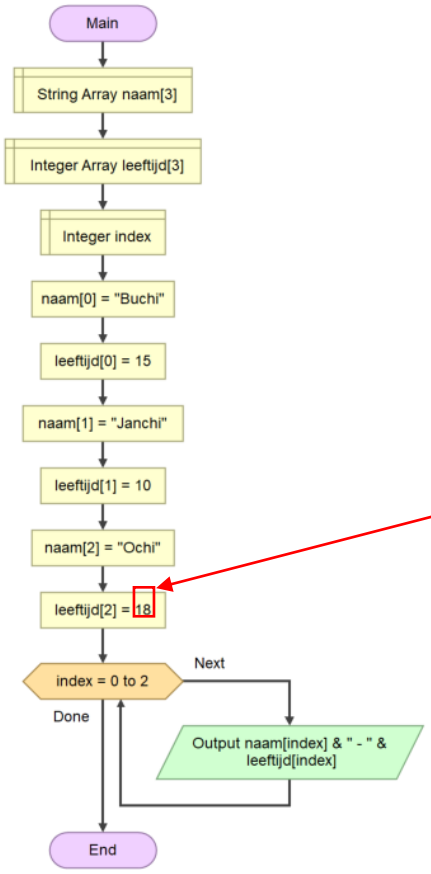


Helaas is dit *(nog)* niet mogelijk. Om deze beperking Overbruggen kunnen parallelle Arrays maken.

Bijvoorbeeld in één kolom staan namen en de andere leeftijden. In Flowgorithm zou dit er zo uit zien:



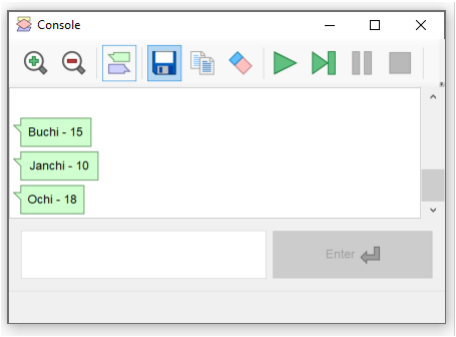
Flowgorithm



Twee verschillende arrays waar de indexen gelijk moeten lopen om een twee dimensionale array te simuleren.

Lijsten doorlopen
Wanneer je alles van een lijst wil laten zien is het beste om een **for**-lus te gebruiken, waar bij de **for**-lus telt van nul tot het aantal elementen van de array min één, als deze zero-based is

Aantal elementen is 3, in de **for**-lus wordt dat dus 3 – 1 = 2



Beperkingen:

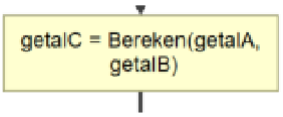
In Flowgorithm is het helaas niet mogelijk om een Array variabele als return waarde mee te geven in een functie. Je kunt een Array variabele wel meegeven als parameter. Deze wordt dan gezien als een referentie Flowgorithm (Reference) variabele en niet een **waarde** (Value) variabele wat bij parameters gebeurt die niet van het type Array zijn.

Een parameter dat een waarde variabele is, krijgt alleen de waarde mee van de doorgegeven variabele, welke op een nieuwe geheugenlocatie tijdelijk wordt opgeslagen door de computer. De waarde in de originele variabele blijft onveranderd. Het is niet mogelijk om een variabele die in **Main** is gedeclareerd in een functie aan te passen.

Flowgorithm

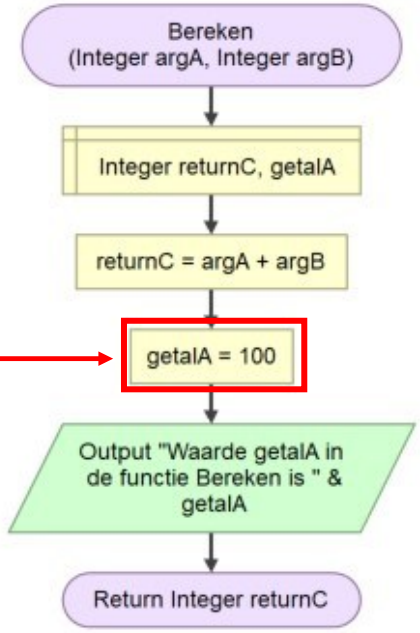
Bekijk de volgende functie **Bereken**:

In de functie **Bereken** wordt er vanuit **Main** de waarde van getalA en getalB meegegeven aan de functie Berkenen in de parameters **argA** en **argB**.



Echter wordt er ook een **getalA** gedeclareerd en een waarde toegekend in de functie.

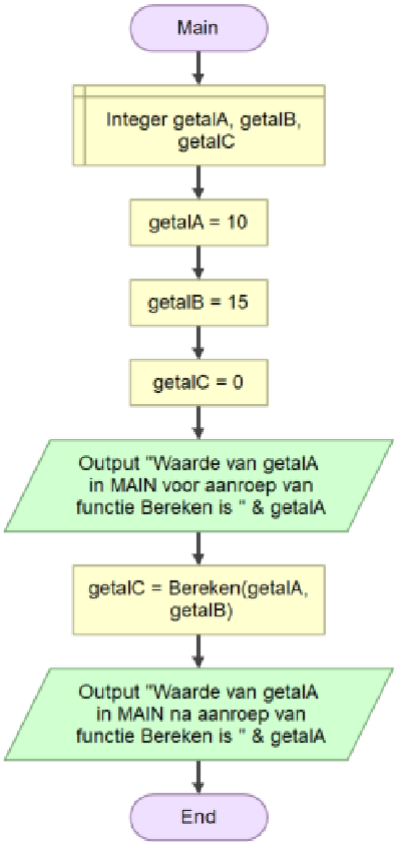
Nadat de functie **Bereken** is uitgevoert wordt de waarde van **getalA** in **Main** weer afgedrukt.



Merk op dat de waarde van **getalA** in **Main** **onveranderd** is gebleven nadat de functie **Bereken** is uitgevoerd. Alleen de waarde van **getalA** in **Main** is meegegeven aan de functie. De variabele **getalA** in de functie **bereken** ***is een totaal andere variabele*** die de Flowgorithm op een andere plek in het geheugen heeft opgeslagen.

Flowgorithm

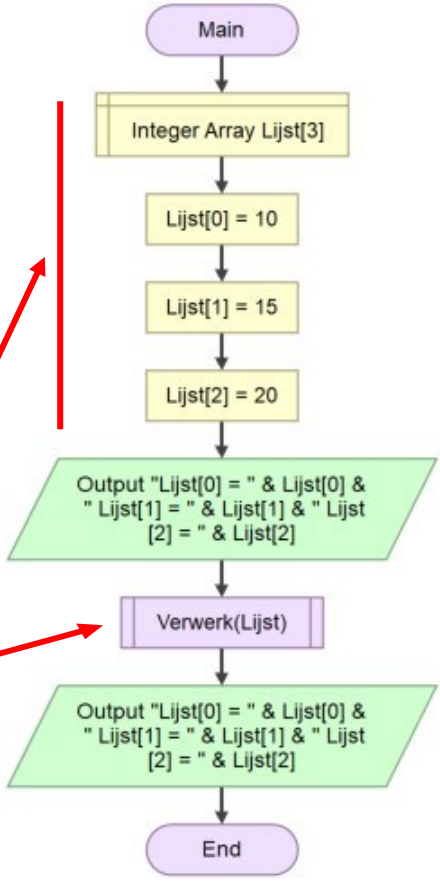
In de afbeelding hieronder is het stroomdiagram van Min te zien.



Bij een **referentie** variabele/parameter wordt **niet de waarde(n)** van de variabele meegegeven **maar het adres** waar deze waarde(n) te vinden zijn in het interne geheugen van een computer.

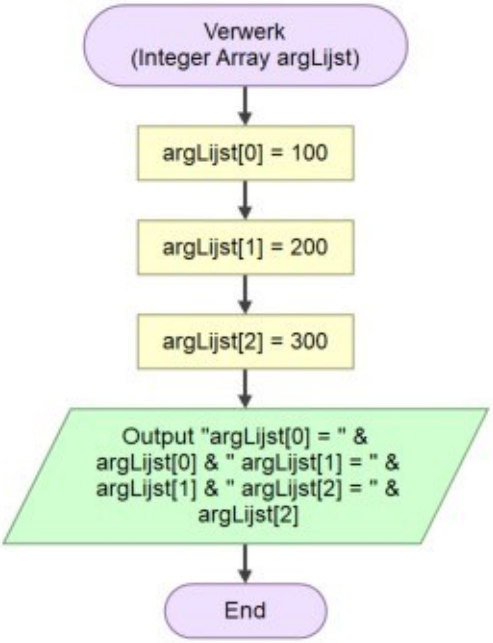
In het stroomdiagram hiernaast wordt in de **Main** een **Lijst** gedeclareerd van drie elementen en de elementen worden geïnitialiseerd.

De **Lijst** wordt als parameter eegegeven aan de functie **Verwerk**.

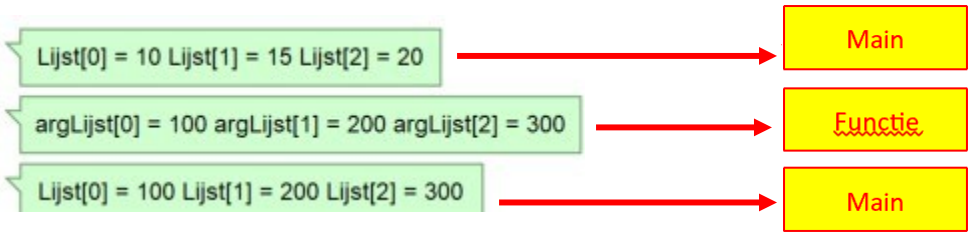


Flowgorithm

In de functie Verwerk worden de waarden van de elementen gewijzigd, waarna in **Main** de waarden van **Lijst** opnieuw worden afgedrukt.



Dit geeft als resultaat de vlgende uitvoer:

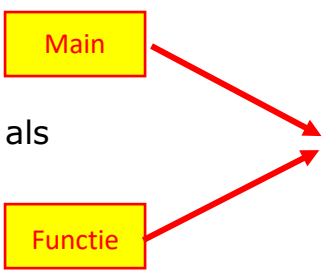


Merk op dat de waarden van de elementen van Lijst **veranderd zijn na dat** de functie Verwerk is uitgevoerd. Dit komt omdat een Array variabele als ***referentie*** wordt meegegeven als parameter.

Dit betekent dat **het adres** waar de waarden van de Array zijn opgeslagen wordt meegegeven. Als dan een waarde wordt veranderd geldt dit ook voor buiten de functie.

Daar de Array variabele **in Main** en de parameter variabele **in de functie** beiden naar **hetzelfde stuk interne geheugen** kijken.

Dus als de waarde in dat stuk geheugen wordt gewijzigd voor *de ene* (in Main) variabele, dan geldt de wijziging ook voor *de andere* variabele (in de Functie).



Interne geheugen	
Index	Waarden
[0]	10
[1]	20
[2]	30

Conclusie is dat indien je waarden in een parameter Array wil wijzigen je deze niet return waarden in een functie hoeft op te geven.

Flowgorithm

Ingebouwde functies *(Intrinsic Functions)*

Flowgorithm kent een aantal ingebouwde functies die gebruikt kunnen worden in programma's. Op de website van Flogorithm ga naar Documentation, daar staat onder het kopje *Expressions* de optie om de verschillende ingebouwde functies te bekijken.

Als je op de optie *Intrinsic Functions* klikt krijg je een nieuwe pagina met tabellen waarin alle functies beschreven worden, verdeeld naar hun categorie.

Mathematics

Function	Description
`Abs(n)`	Absolute Value
`Arcsin(n)`	Trigonometric Arcsine
`Arccos(n)`	Trigonometric Arccos
`Arctan(n)`	Trigonometric Arctangent
`Cos(n)`	Trigonometric Cosine
`Int(n)`	Integral (whole value) of a real number
`Ln(n)`	Natural Log
`Log(n)`	Natural Log (same as Ln)
`Log10(n)`	Log Base 10
`Sgn(n)`	Mathematical sign (-1 if ***n*** is negative, 0 if zero, 1 if positive)
`Sin(n)`	Trigonometric Sine
`Sqrt(n)`	Square Root
`Tan(n)`	Trigonometric Tangent

Strings

Function	Description
`Len(s)`	Length of a string
`Char(s, i)`	Returns a character from the string ***s*** at index ***i***. Characters are indexed starting at 0.

Data Type Conversion

Function	Description
`ToChar(n)`	Convert a character code ***n*** into a character.
`ToCode(c)`	Convert a character ***c*** into a character code (integer).
`ToFixed(r, i)`	Convert real number ***r*** to a string with ***i*** digits after the decimal point. This function is useful for currency.
`ToInteger(n)`	Convert a string to an integer
`ToReal(n)`	Convert a string to an real
`ToString(n)`	Convert a number to a string

Other

Function	Description
`EOF()`	Returns true if the end of the file was reached. This is used with files opened for reading.
`Random(n)`	A random number between 0 and (***n*** - 1)
`Size(a)`	The size (number of elements) in an array

Er is eerder al kennis gemaakt met de functie **Random()** om een willekeurig getal te laten genereren door de computer.

Hier onder volgen een aantal voorbeeld programma's voor het gebruik van een aantal van de functies.

Opdracht 12

Schrijf een programma dat de absolute waarde geeft van door de gebruiker ingevoerde getallen en stopt wanneer de gebruiker het cijfer nul als waarde opgeeft.

Opdracht 13

Schrijf een programma dat de gehele waarde geeft van een decimaal (real) getal. Het programma stopt wanneer de gebruiker het cijfer nul als waarde opgeeft.

Opdracht 14

Schrijf een programma dat de wortel berekend van een positief getal dat door de gebruiker wordt ingevoerd. Het programma stopt wanneer de gebruiker het cijfer nul als waarde opgeeft.

Opdracht 15

Schrijf een programma dat de lengte bepaald (aantal tekens) van een tekst dat door de gebruiker wordt ingevoerd. Het programma stopt wanneer de gebruiker niks invoert maar op <ENTER> of <RETURN> knop drukt.

Opdracht 16

Schrijf een programma dat het teken op een bepaalde positie in een tekstregel toont. Het programma stopt wanneer de gebruiker niks invoert maar op <ENTER> of <RETURN> knop drukt.

Opdracht 17

Schrijf een programma dat het aantal spaties telt in een door de gebruiker ingevoerde tekst. Het programma stopt wanneer de gebruiker niks invoert maar op <ENTER> of <RETURN> knop drukt.

Opdracht 18

Schrijf een programma dat controleert of een door de gebruiker gekozen woord in een tekstregel voorkomt en dan de begin positie van het woord in die tekstregel aan geeft.
Het programma stopt wanneer de gebruiker niks invoert maar op <ENTER> of <RETURN> knop drukt.