

1 Programmeren: praten met je computer

© Rik Palmans

Gewoon Nederlands praten met een computer is voorlopig nog toekomstmuziek. Maar toch is het mogelijk een computer bevelen te geven. Je typt je bevelen in op het toetsenbord, in een taal die aan strenge regels (de ‘syntax’ van de programmeertaal) is gebonden. De computer moet ook over een systeem beschikken dat in staat is deze bevelen in mensentaal om te zetten in echte machinetaal (codes in de vorm van binaire getallen die door de microprocessor worden begrepen). We maken onderscheid tussen twee vertaalsystemen: **compilers** en **interpreters**, maar in deze cursus gaan we daar niet dieper op in.

Er bestaan tientallen programmeertalen en geen mens kent die allemaal. Maar de basisprincipes blijven altijd hetzelfde. Als je eenmaal één taal hebt geleerd, is het nadien veel makkelijker om een volgende te leren. De taal waarmee jullie voorgangers hier op school leerden werken, heet **HyperTalk**. Ze maakt deel uit van het pakket **HyperCard**. Toen dat programma in 1987 op de markt kwam, betekende het een enorme stap voorwaarts. Het bevatte een aantal vernieuwende concepten; zo was HyperCard het eerste pakket waarmee je *hyperlinks* kon maken en het inspireerde daardoor de ontwikkelaars van het WWW. De programmeertaal HyperTalk diende dan weer als inspiratiebron voor de ontwikkelaars van JavaScript, een in webpagina’s vaak gebruikte programmeertaal.

Helaas stopte Apple enkele jaren later met de ondersteuning van HyperCard en kwamen er geen nieuwe versies meer uit. Maar diverse andere ontwikkelaars namen de draad op en maakten HyperCard-klonen. De meest recente in dat rijtje is **revMedia**, met als programmeertaal **revTalk**.

De ontwikkelaars van dit pakket hebben, net als bij HyperCard en ook Powerpoint, als metafoor een **stapel** — een *stack* — **kaarten** genomen. Wat je op het scherm ziet als je een bestand opent, is een ‘kaart’. Aan de rechterkant van het scherm heb je geen schuifbalk zoals in de meeste programma’s. Om de rest van het bestand te zien, moet je naar een andere kaart gaan.

RevMedia is zowel voor Macintosh, Windows als Linux beschikbaar. Dat was voor ons een bijkomende reden om over te stappen van HyperCard naar revMedia. Ten opzichte van HyperCard is revMedia een stuk krachtiger, maar dus ook complexer. Het is daarom belangrijk dat je alle toeters en bel- len die we in de diverse oefeningen toch niet gebruiken, zeker in het begin links laat liggen. Als je eventueel de smaak van het programmeren te pakken krijgt, kun je je daar later nog in verdiepen.

Voor meer informatie over revMedia: <http://www.runrev.com>. Je kunt er onder meer revMedia gratis downloaden. Het softwarebedrijf dat revMedia maakt, bracht, brengt ook krachtigere versies ervan op de markt; daarvoor moet je wel betalen. Op deze site vind je ook allerhande ondersteunend ma- teriaal.

2 RevMedia werkt met objecten

Met revMedia werken is voor een stuk knutselen, te vergelijken met het maken van een collage. De bouwstenen waar we mee werken noemen we **objecten**. De basisobjecten zijn de stack en de afzonderlijke kaarten (dus het scherm dat je voor je hebt). Op zo'n kaart kun weer je diverse andere **objecten** plakken.

De belangrijkste zijn velden en knoppen. In de regel gebruik je

velden: om gegevens in te voeren (input) of resultaten te tonen (output);

knoppen: om instructies mee te geven.

Het kan ook anders maar daar gaan we voorlopig niet dieper op in.

Daarnaast kun je ook **grafische objecten** invoeren of maken met behulp van het tekengereedschap in het palet. Eventueel kunnen die zich ook als knoppen gedragen.

De **instructies** die een object kan uitvoeren, zitten in het **script** dat deel uitmaakt van het object. Een instructie in revtalk bevat altijd een werkwoord in de bevelende vorm, 'put' bijvoorbeeld. De rest van de zin moet strikt aan de voorgeschreven **syntax** (taalregels) beantwoorden. Zo mag je niet zeggen "put ... *in* fld ...". Het moet wel degelijk '*into*' zijn — anders krijg je een foutmelding. Een overzicht van welke instructies mogelijk zijn, kun je opvragen in de *Dictionary* (op te roepen via de *toolbar* of het Help-menu).

De ontwerpers van revtalk hebben geprobeerd deze programmeertaal zo dicht mogelijk bij gewoon Engels te doen aansluiten. Dat is een voordeel voor niet-professionele programmeurs. (Professionele programmeurs vinden dat eerder een nadeel omdat zo'n programma een stuk langer is. Zij willen vooral een programmeertaal die zo compact mogelijk is.)

Je mag in een programma overigens ook gewone (Nederlandse) tekst opnemen, als commentaar voor jezelf. Om de computer duidelijk te maken dat die regels geen deel uitmaken van het eigenlijke programma, laat je ze voorafgaan door '--', of je plaatst ze tussen '/**' en '*/'.

Opdrachten

1. Maak een nieuwe *mainstack*. Bewaar die in je eigen map op de harde schijf.

De structuur van een stack kan naast een mainstack ook diverse substacks omvatten. Voorlopig gaan we daar we daar niet dieper op in.

2. Maak op de eerste kaart **oefening 1**. Dat houdt in dat je er een veld en een knop op plaatst.

Noteer hoe je die objecten maakt, op de gewenste plaats op het scherm zet en grootte en vorm verandert. Ook belangrijk is dat je ze de naam geeft die in de uitleg over de oefening is aangegeven. Je mag eventueel zelf andere namen kiezen, maar je moet er dan wel aan denken in het script exact dezelfde namen te gebruiken.

3. Vervolgens plaats je in het **script** van de knop de instructie die de gevraagde opdracht uitvoert.

4. Lees als achtergrondinformatie bij dit hoofdstuk de teksten:

- Wat zijn objecten?
- Scripts

*De inhoud van deze teksten moet je **kennen**. Dat wil zeggen: je moet er op een toets vragen over kunnen beantwoorden zonder dat je de syllabus en/of je eigen notities ter beschikking hebt. Bij het maken van oefeningen mag je die wel gebruiken.*

5. Maak in de stack waarin je oefening 1 hebt gemaakt, een nieuwe kaart. Zowel op de eerste als op de tweede kaart moeten knoppen staan om makkelijk naar de volgende respectievelijk vorige kaart te gaan. Daarvoor is het nodig dat je eerst (dus vooraleer je de nieuwe kaart maakt!) de hieronder beschreven handelingen uitvoert:

- Maak een nieuwe knop. Geef (via de Inspector, op te roepen via de *toolbar* of door te dubbelklikken op een object) deze knop als *icon* (symbool) een naar links wijzende pijl. In het script van de knop moet je de juiste instructie plaatsen. *Noteer de verschillende mogelijkheden waarop je de scripts van deze knoppen kunt noteren.*
- Kopieer deze knop, plak hem, sleep hem naar de gewenste positie en pas vervolgens in de Inspector *icon* en *script* aan.

- **Gropeer** de knoppen.
- Vink in de Inspector de keuze “Behave like a background” aan.
Daardoor zal bij het maken van een nieuwe kaart automatisch al dit (gegroepeerde) object op die kaart worden geplaatst.

Houd er rekening mee dat een dergelijk gegroepeerd background-element bij wissen meteen van alle kaarten wordt verwijderd!

6. Maak op de tweede kaart **oefening 2**.

Extra-info 1: Wat zijn objecten?

In object-georiënteerde programmeeromgevingen (OOP's) worden toepassingen opgebouwd met behulp van objecten. Zonder te vervallen in al te abstracte definities, kun je stellen dat je met objecten te maken hebt als ze volgende drie kenmerken bezitten:

- Ze hebben **eigenschappen** (properties), op grond waarvan ze tot een bepaalde 'klasse' behoren.

Een vergelijking om duidelijk te maken wat een klasse is: een hond. De hele 'honden-objectenklasse' vertoont een aantal eigenschappen waaraan we de leden ervan als 'hond' herkennen: ze blaffen, ze rennen kwispelstaartend en bomenplassend door het leven, ze komen met smekende ogen naast je zitten als je een koekje aan het eten bent, ...

Een klasse kun je weer indelen in subklassen. Alle subklassen hebben de kenmerken 'geërfd' die gemeenschappelijk zijn voor de hele klasse, maar elk hebben ze ook hun eigen specifieke kenmerken. Zo is een Duitse herder een stevig uit de kluiten gewassen knaap met een typische vorm en kleur, die hem duidelijk van een poedel onderscheidt.

- Ze kunnen **berichten ontvangen** (en zenden), die bij de ontvanger een in een 'verwerker' vastgelegde reactie losmaakt.

Als je bijvoorbeeld 'af' zegt tegen een hond, zal hij daarop reageren — tenminste als er bij hem een 'verwerker' voor dat 'bericht' aanwezig is. Met andere woorden: als de hond in kwestie afgericht — geprogrammeerd — is om op dat bericht te reageren.

Maar vermits de inhoud van die 'verwerker' niet noodzakelijk voor elke hond hetzelfde is, kan eenzelfde bericht bij verschillende dieren ook verschillende reacties teweeg brengen. De eerste hond zal misschien gaan liggen als hij het bevel 'af' krijgt, terwijl een andere een voorwerp loslaat dat hij op dat ogenblik vastheeft.

- Objecten kunnen op hun beurt **andere objecten bevatten**.

Om bij onze hond te blijven: de staart van de hond is ook een object (hoewel ik me niet meteen kan inbeelden dat hij op bevelen reageert).

RevMedia is weliswaar geen volwaardige object-georiënteerde programmeeromgeving (OOP) omdat je er zelf geen nieuwe klassen, met nieuwe eigenschappen, in kunt maken. Maar met de ingebouwde objecten kom je al een heel eind.

De belangrijkste objecten in revMedia zijn:

- de stack zelf;
- de kaarten van die stack;
- velden, knoppen en grafische objecten op een kaart.

De eigenschappen van die objecten kun je veranderen via een dialoogvenster. Dat roep je op door een object te selecteren en onder 'Object' 'Object Inspector' te kiezen. Een kortere weg is: dubbelklikken op een object.



Om een object te kunnen selecteren en er eventueel de eigenschappen van te veranderen, moet je in de 'Tools Palette' de 'pointer' (rechts bovenaan) kiezen. Als je de browser (links bovenaan) kiest, kun je de stack gebruiken. Je kunt deze keuze ook maken via het Tools-afrolmenu: Browse Tool of Pointer Tool.

Objectnamen

Vooraf belangrijk in de programma's die wij zullen maken, is de **naam** van een veld. Dat is één van de eigenschappen die je via de Object Inspector aan een object kunt toekennen. Je kunt dan verwijzen naar een bepaald veld met de omschrijving

field "tekst" (1)

of field "tekst" of card "oefening" (2)

(1) *In plaats van 'field' mag je ook de afkorting 'fld' gebruiken.*

(2) *Het tweede deel "of card..." is maar vereist als het om een veld op een andere kaart gaat. Deze omschrijving geeft wel aan wat we met het derde kenmerk (dat objecten deel kunnen uitmaken van andere objecten: zie vorige pagina) bedoelen.*

Groepen

Je kunt diverse objecten groeperen tot één nieuw object: een 'group'. Belangrijk is dat je een dergelijke groep dan als **achtergrond van een kaart** kunt laten functioneren. Dat houdt in dat die groep automatisch wordt overgenomen als je een nieuwe kaart maakt. Dat is bijvoorbeeld handig als je op elke kaart knoppen wilt plaatsen die je naar de vorige respectievelijk volgende kaart brengen. In de Object Inspector moet je die eigenschap dan aanvinken.

Objecten groeperen doe je door ze allemaal te selecteren — houd de shift-toets ingedrukt terwijl je de diverse objecten achtereenvolgens aanklikt — en dan in de toolbar of in het Object-afrolmenu de instructie *Group (selected)* te geven.

Extra-info 2: Scripts

Je kunt in revMedia een **instructie** geven door de *Message Box* (in het Tools-afrolmenu) op te roepen. Als je daarin een instructie typt en op de Enter- of Return-toets drukt, wordt die onmiddellijk uitgevoerd. Je kunt er ook **expressies** (zie verder) in opgeven, waarvan het resultaat eveneens in de *Message Box* verschijnt.

Maar doorgaans geef je — bijvoorbeeld door op een knop te klikken — meer dan één instructie. Zo'n reeks instructies heet een **programma** en het maken ervan is **programmeren**. Die instructies zitten bij revMedia in het **script** van een object, bijvoorbeeld een knop waarop je kunt klikken.

*Van een **script** spreken we als het programma geen op zich zelf staande toepassing is maar slechts kan draaien binnen een ander programma. Vermits een revTalk-programma maar werkt binnen revMedia zelf, moeten we hier van scripts spreken.*

In een objectgeoriënteerd systeem zijn de instructies verdeeld over een groot aantal kleine programma's in plaats van samen één groot programma te vormen. Elk van die kleine programma's (scripts) maakt deel uit van een object — in revMedia dus van een stack, kaart, knop, veld of afbeelding.

Je kunt het script van een object opvragen via het dialoogvenster van de Object Inspector of door met de rechtermuisknop te klikken op een object. Er opent zich dan een nieuw venster en in de menubalk verandert een en ander: je bevindt je in de *script-editor*. Het is een eenvoudige tekstverwerker om scripts mee te maken en te bewerken.

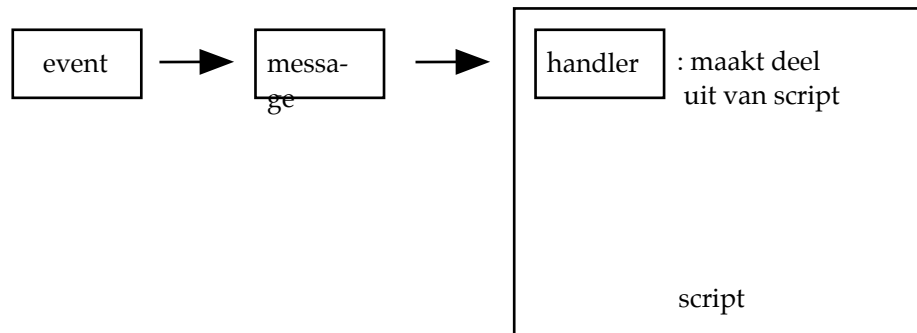
Event driven

Om de werking van een revMedia-stack te begrijpen, moet je weten dat het systeem voortdurend **berichten** (messages) doet circuleren, die signaleren welke **gebeurtenissen** (events) plaatsvinden. Er treedt maar een reactie op wanneer een welbepaalde gebeurtenis plaatsvindt. Zo'n *event* kan bijvoorbeeld het klikken met de muis op een knop zijn.

Een bericht veroorzaakt maar een reactie als er voor dat bericht een **verwerker** (handler) aanwezig is. Die verwerker maakt deel uit van een **script**, dat op zijn beurt deel uitmaakt van een object.

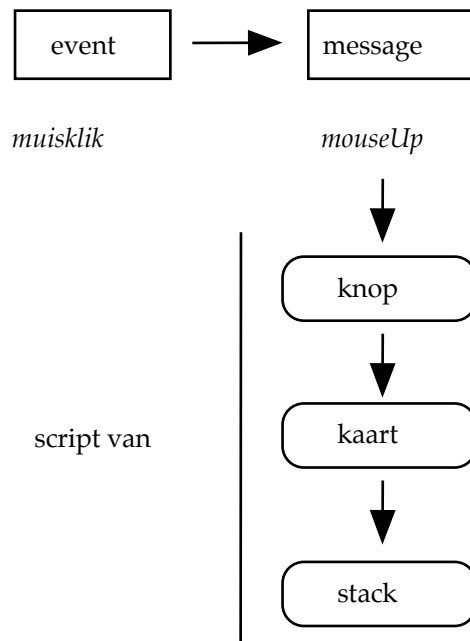
Een voorbeeld:

- event: muisklik (meer bepaald: het loslaten van de muisknop);
- message: mouseUp;
- handler: stukje in script dat begint met "on mouseUp".



Nu kunnen er in een stack meerdere objecten zijn die verwerkers bevatten die beginnen met "on mouseUp". Welke zal dan reageren op de mouseUp-message? Het antwoord is: de verwerker die, tijdens het doorlopen van het hieronder geïllustreerde zoekpad, in contact komt met het bericht (message).

Een bericht volgt een vastgelegde weg, een 'zoekpad':



Het bericht wordt altijd eerst opgevangen door het object waarin de *event* zich voordoet. Dat is bijvoorbeeld de knop waarop je klikt. Als die knop een script heeft met een mouseUp-verwerker, wordt die uitgevoerd. Als dat niet zo is, wordt het bericht doorgegeven naar de kaart waarvan de knop deel

uitmaakt. Dan zoekt het systeem daar naar een mouseUp-verwerker. Eventueel gaat het bericht nog verder tot het op zijn pad zo'n verwerker vindt... of het gaat gewoon verloren omdat het die nergens vindt.

Als je objecten hebt gegroepeerd tot een 'achtergrond', komt een dergelijke achtergrond in het zoekpad achter een kaart.

De meeste berichten worden gegenereerd door handelingen die je met de muis verricht: klikken op een knop of veld, openen of sluiten van een venster, stapel of kaart, ... Zelfs als er niets gebeurt, circuleert er een bericht: 'idle' — 'alles rustig'.

Let ook eens op de manier waarop in een script de regels inspringen. Daar zorgt de script-editor automatisch voor. Als dat inspringen niet gebeurt zoals verwacht, is dat een eerste aanwijzing dat er iets fout is in de syntax van je script. (Druk eventueel op de TAB-toets om alle inspringingen in een script op te roepen.)

3 Variabelen en andere containers

In de scripts die we tot nu toe gemaakt hebben, kwam vaak de `put`-instructie voor. Algemeen heeft die de vorm van

```
put <iets> into <bestemming>
```

Constante

In `<iets>` kan een welbepaalde waarde staan, het getal 5 bijvoorbeeld. We spreken dan van een ‘constante’ — meer bepaald een numerieke constante in dit geval. Het kan ook een alfanumerieke constante, een ‘string’, zijn. In `revTalk` geef je dat aan door de inhoud ervan tussen (dubbele) aanhalingstekens te plaatsen:

```
put "school" into <bestemming>
```

Expressie

In `<iets>` kan ook het resultaat van een expressie of bewerking staan. Als je bijvoorbeeld schrijft:

```
put (5+3) into <bestemming>
```

dan zal het getal 8 in `<bestemming>` worden geplaatst.

Net zoals in *spreadsheets* zijn expressies opgebouwd uit numerieke of alfanumerieke waarden, operatoren en functies.

Expressies zijn algemeen: alle uitdrukkingen die een resultaat opleveren. Dat kunnen wiskundige bewerkingen zijn maar ook bewerkingen met tekststukjes (die we ‘alfanumerieke gegevens’ noemen). Wellicht ken je het gebruik van expressies al uit spreadsheetprogramma’s als Excel: daar waren het celinhouden die beginnen met het `=`-teken. Expressies bevatten constanten en/of variabelen; de bewerkingen daarop worden aangegeven door operatoren (de bewerkingstekens uit de wiskunde: `+` `-` `*` `/`) en/of functies.

Functie

Een voorbeeld van een functie is

```
put sqrt(25) into <bestemming>
```

wat de vierkantswortel van 25 oplevert. Maar je mag ook schrijven:

```
put the sqrt of 25 into <bestemming>    -- let op het woordje the!
```

`RevMedia` kent een aantal ingebouwde functies, maar zoals we verder zullen zien kun je ook zelf functies maken. Een functie bestaat altijd uit een functienaam, gevolgd door één of meer **argumenten**. Zoals je in het voorbeeld hierboven ziet, zet je die argumenten tussen haakjes achter de functienaam. (Als het er meerdere zijn, moet je ze van elkaar scheiden met een komma.) Voor de ingebouwde functies (die overigens altijd maar één argument hebben)

kun je ook kiezen voor de tweede formulering, die meer aansluit bij de manier waarop we het ook in het dagelijks leven formuleren: “de vierkantswortel van 25”.

Als je het resultaat van een expressie wilt kennen, kun je die in de **Message Box** (berichtvenster) typen. Het resultaat verschijnt in hetzelfde venster. Dat berichtvenster kun je ook gebruiken om één of enkele instructies aan een revMedia-stack te geven. Dat kan handig zijn als je een onderdeel van een script even wilt uitproberen.

Velden

In de put-instructie die we hierboven als voorbeeld gaven, heb je bij <bestemming> waarschijnlijk meteen gedacht aan een **veld**. We hebben immers al oefeningen gemaakt waarin we die gebruikten.

Velden kunnen ook onzichtbaar zijn. Dat kan handig zijn om bijvoorbeeld gegevens bij te houden die je niet voortdurend op het scherm wilt tonen.

Variabelen

Heel vaak heb je in een programma een situatie waarin je resultaten van bewerkingen slechts tijdelijk moet kunnen bijhouden. Voor die situatie bestaat een elegante oplossing: je slaat ze op in **variabelen**. Je kunt ze beschouwen als denkbeeldige, onzichtbare velden.

Een variabele is een plaats in het RAM-geheugen van de computer, die je met een put-instructie een waarde kunt geven zoals je dat doet met een veld. Alleen moet je zelf bijhouden welke waarde erin zit; je ziet de inhoud ervan immers niet op het scherm. (Er bestaat wel een manier om de inhoud van variabelen zichtbaar te maken, maar daar gaan we hier niet dieper op in.)

Je maakt een variabele door in een put-instructie een naam te gebruiken die zeker niet voorkomt in de revTalk-woordenschat. Als je bijvoorbeeld schrijft: `put sqrt(25) into uitkomst`

dan wordt het getal 5 opgeslagen in een geheugenplaats die even de naam ‘uitkomst’ krijgt. Even, inderdaad: als het programma — de *handler* — is uitgevoerd, wordt die geheugenplaats weer leeggemaakt en bestaat de variabele niet langer. *We zullen later uitleggen hoe je er toch voor kunt zorgen dat een variabele langer blijft bestaan.*

Als je ergens in een programma een nieuwe naam vermeldt, gaat revTalk ervan uit dat je daarmee een variabele bedoelt. *Let wel op: die naam mag niet tussen aanhalingstekens staan, zoals bij de naam van een veld.*

Houd ook rekening met de volgende opmerkingen bij het geven van een naam aan een variabele:

1. Alleen namen die geen andere betekenis in revTalk, zijn toegelaten. Als je Nederlandse namen neemt, zit je dus veilig.
2. De naam moet met een letter beginnen; verder in de naam zijn wel cijfers toegelaten. revTalk maakt geen onderscheid tussen kleine en hoofdletters.
3. Tekens die niet zijn toegelaten zijn onder meer: spatie, deelstreepje (of andere tekens die als operator in expressies worden gebruikt). Wél toegelaten: het onderlijnstreepje, zoals in 'mijn_naam'. Speel liefst op veilig en beperk je in variabelenamen tot gewone letters (geen diakritische lettertekens), cijfers en het onderlijnstreepje. Gebruik ook zoveel mogelijk namen die aangeven over welke waarde het gaat. Dat maakt een script een heel stuk leesbaarder.

Containers

Elke structuur waarin je in een revMedia-stack gegevens kunt opslaan en bewerken, noemen we een container. We hebben al drie soorten containers ontmoet:

- velden;
- variabelen;
- het berichtvenster: daarin kunt je een waarde stoppen met
`put sqrt(25) into message` (afgekort: `msg`)
of gewoon `put sqrt(25)`

Chunks

Een speciale eigenschap van revTalk is dat je ook met stukken van containers kunt werken. Zo'n stuk of *chunk* gedraagt zich ook als een volwaardige container. Vooral bij teksten zijn ze enorm handig. Voorbeelden van chunks zijn *character* (afgekort: *char*), *word*, *line* en *item*. Je kunt dus schrijven:

```
put "voorbeeld" into word 2 of fld "Tekst"
```

Het tweede woord van de tekst in het bedoelde veld wordt dan vervangen door 'voorbeeld'.

4 Algoritmen

Je geeft instructies aan een computer niet één voor één, maar als een hele reeks waarin alle handelingen zitten vervat die je de computer wilt laten uitvoeren. Zoals we in het vorige hoofdstuk hebben gezien, vormt een dergelijke reeks instructies een verwerker.

Zo'n verwerker is een voorbeeld van een **algoritme**. Algemeen is dat een beschrijving die aangeeft hoe je vanuit een gegeven beginsituatie een bepaald resultaat kunt bereiken. Ook heel wat processen in het dagelijks leven zijn als algoritmen te beschrijven. Denk maar aan het klaarmaken van een gerecht: het recept is een algoritme dat beschrijft hoe je met de diverse ingrediënten iets lekkers kunt bereiden.

Niet alle processen in het dagelijks leven zijn algoritmisch opgebouwd. Het schrijven van een boek bijvoorbeeld laat de menselijke verbeelding werken, en die is niet in algoritmen te vangen.

Een computerprogramma is dus niets anders dan een in een specifieke programmeertaal beschreven algoritme. De deelhandelingen van een algoritme kunnen we als bouwstenen beschouwen, die aan elkaar worden vastgevoerd. In de structuur die de bouwstenen met elkaar verbindt, zijn combinaties van de volgende drie basisschema's mogelijk:

- sequentie;
- selectie (keuze);
- iteratie (herhaling).

Sequentie

Als de instructies in een algoritme zonder meer uit een reeks op elkaar volgende stappen bestaan, spreken we van een **sequentie**, zoals in volgend voorbeeld:

```
on mouseUp
  beep
  hide me
  wait 3 seconds
  beep
  show me
end mouseUp
```

Elke regel is een instructie (te herkennen aan het werkwoord in de bevelende vorm). Probeer maar eens uit wat bovenstaande verwerker precies doet door hem in het script van een knop of een grafisch object te plaatsen.

Selectie

De uitkomst van het bovenstaande script zal altijd hetzelfde zijn. Maar er zijn situaties waarin een programma een **keuze** moet kunnen maken. Een voorbeeld vind je in **oefening 3**.

Opdrachten

1. Maak oefening 3. In eerste instantie is dat de eenvoudige vorm: zonder controle of er niet per ongeluk een spatie of return is getypt.
2. In dit script maak je gebruik van zaken die in het vorige hoofdstuk werden besproken, zoals variabelen. Noteer, aan de hand van concrete voorbeelden, welke begrippen in dit script aan bod zijn gekomen.

Het script van de knop in oefening 3 bevat een **selectie**: het programma kan hier twee verschillende zaken doen. Welke van de twee het kiest, hangt af van het resultaat van een **booleaanse expressie**. Als die de waarde 'true/ waar' oplevert, wordt de eerste instructie uitgevoerd, anders — 'else' — de tweede.

Wat is een booleaanse expressie?

Een booleaanse expressie is een bewerking waarvan de uitkomst 'true' of 'false' is. Je kunt het zien als een alternatieve formulering van een vraag waarop slechts de antwoorden 'ja' en 'nee' mogelijk zijn. Een voorbeeld uit het dagelijks taalgebruik: "Is Brussel de hoofdstad van België?" luidt in een booleaanse formulering: "Brussel is de hoofdstad van België." Het antwoord op de vraag is 'ja'; de booleaanse expressie levert dan als uitkomst 'true'. Een ander voorbeeld: "Antwerpen is de hoofdstad van Nederland" levert de uitkomst 'false' op.

Booleaanse expressies hebben hun eigen operatoren (= < > <> AND NOT).

Het kan zijn dat het programma dat je gemaakt hebt voor de eerste opdracht op de voorgaande pagina aangeeft dat de twee woorden verschillend zijn, hoewel je op het scherm hetzelfde woord ziet. Waarschijnlijk staat achter één van de twee woorden dan een spatie of een return-teken. Je kunt het script uitbreiden met instructies die testen of dergelijke tekens in de velden voorkomen, maar bij een eerste poging laat je dat best nog even achterwege. (Daarvoor moet je eerst nog leren wat een iteratie is; dat bespreken we dadelijk.) Let dus even op dat je dergelijke onzichtbare tekens niet per ongeluk in de velden invoert.

Je kunt bij een selectie ook de keuze laten maken uit meer dan twee mogelijkheden. Hoe je de selectie dan moet formuleren, zie je in het volgende voorbeeld:

```
on mouseUp
  put fld "keuze" into getal
  if getal = 1 then
    put "een"
  else if getal = 2 then
    put "twee"
  else if getal = 3 then
    put "drie"
  else if getal = 4 then
    put "vier"
  else if getal = 5 then
    put "vijf"
  end if
end mouseUp
```

Iteratie

Bij een iteratie of herhaling wordt een deel van het algoritme meerdere keren na elkaar uitgevoerd. Je kunt dan twee mogelijkheden onderscheiden:

- **begrensde iteratie:** je weet op voorhand hoeveel keren dat moet gebeuren;
 - **voorwaardelijke iteratie:** je weet dat niet. In dat geval moet je een selectie inbouwen, waarin je de vraag laat stellen of de 'lus' nog een keer moet worden doorlopen.
-

Opdrachten

1. Probeer enkele (minstens drie) voorbeelden te bedenken van iteraties in het dagelijks leven. Om ze met een programmeerinstructie te kunnen vergelijken, moet je ze telkens als een **bevel** formuleren.
2. Vul de voorgaande oefening (twee woorden vergelijken) aan zodat gelijke woorden worden herkend ook als de gebruiker per ongeluk een spatie, return of ander onzichtbaar teken heeft getypt.
3. Maak oefening 4.
4. Hierin gebruik je een iteratie. Van welk type? Leg uit *waarom* je hier voor dat type kiest.
5. Maak oefening 5.
6. Van welk type iteratie maak je hier gebruik? Leg ook hier uit *waarom* je voor dat type kiest.
7. Noteer zowel bij oefening 4 als 5, welke begrippen uit het vorige hoofdstuk aan bod komen.

Extra-info 3: Iteraties

Iteraties of herhalingen zijn het meest complexe onderdeel van algoritme-bouwstenen omdat je ze op diverse manieren kunt formuleren. Zoals we hierboven al aangaven, kan een iteratie begrensd of voorwaardelijk zijn.

Begrensd

De eenvoudigste manier om een **begrensd** iteratie te formuleren is zoals in het volgende voorbeeld:

```
on mouseUp
  repeat 5 times
    beep
  end repeat
end mouseUp
```

Op de tweede regel mag je zelfs 'times' weglaten.

repeat with ...

Een andere formulering maakt gebruik van een variabele (die we hier 'i' hebben genoemd). Je dient dan de beginwaarde en de eindwaarde ervan op te geven. Bij elke herneming neemt de waarde van i met 1 toe. De herhaling stopt wanneer de variabele de opgegeven eindwaarde heeft bereikt.

```
on mouseUp
  put 5 into eindwaarde
  repeat with i = 1 to eindwaarde
    beep
    wait i seconds
  end repeat
end mouseUp
```

In plaats van de teller telkens met 1 te laten toenemen, kun je hem ook met 1 laten afnemen:

```
on mouseUp
  put 5 into beginwaarde
  repeat with i = beginwaarde down to 1
    beep
    wait i seconds
  end repeat
end mouseUp
```

Voorwaardelijk

Bij **voorwaardelijke** iteraties moet je altijd een **booleaanse expressie** inbouwen. Er bestaan twee varianten:

repeat while ...

Als de uitkomst van de booleaanse expressie 'true' is, wordt de iteratie hernomen; als ze 'false' is, stopt de iteratie:

```
on mouseUp
  put 1 into i
  repeat while i ≠ 5
    beep
    add 1 to i
  end repeat
end mouseUp
```

repeat until ...

Juist omgekeerd: als de booleaanse expressie 'false' is, wordt de iteratie her-nomen; bij 'true' stopt ze:

```
on mouseUp
  put 1 into i
  repeat until i = 5
    beep
    add 1 to i
  end repeat
end mouseUp
```

De booleaanse expressie kan ook als een selectie (if ... then ... else) in het script zijn opgenomen:

```
on mouseUp
  put 1 into i
  repeat
    beep
    add 1 to i
    if i < 5 then
      next repeat
    else
      exit repeat
    end if
  end repeat
end mouseUp
```

Een fout die je vaak aantreft in een iteratie is de '**oneindige lus**': er wordt nooit een stopsignaal bereikt, zodat het programma(onderdeel) zich maar blijft herhalen. Als je in één van je scripts een dergelijke fout vaststelt, kun je aan de noodrem trekken met de toetsencombinatie Command-: of Command-Option-Escape. In Windows moet je de Control-toets indrukken in plaats van de Command-toets.

In het bovenstaande voorbeeld zou je iets dergelijks krijgen als je de instructie 'add 1 to i' zou vergeten.

Extra-info 4: Eigenschappen van objecten

We kennen intussen al de objecten waaruit een stack is opgebouwd: velden, knoppen, kaarten, ... Al deze objecten en ook andere elementen als afbeeldingen hebben bepaalde **eigenschappen** (properties).

Een voorbeeld:

```
on mouseUp
  put the location of fld "Tekst" into line 1 of fld "Tekst"
  put the textFont of fld "Tekst" into line 2 of fld "Tekst"
  put the textStyle of fld "Tekst" into line 3 of fld "Tekst"
  put the textHeight of fld "Tekst" into line 4 of fld "Tekst"
  put the lockText of fld "Tekst" into line 5 of fld "Tekst"
end mouseUp
```

Location (positie), textFont (lettertype), textHeight (regelafstand) en lockText (het is al dan niet mogelijk in het veld te typen) zijn eigenschappen van een veld. Je kunt die instellen via een dialoogvenster maar je kunt ze ook via instructies in een script veranderen:

```
on mouseUp
  set the location of fld "Tekst" to "250,220"
  set the textFont of fld "Tekst" to "Times"
  set the textStyle of fld "Tekst" to "bold"
  set the textHeight of fld "Tekst" to 24
  set the lockText of fld "Tekst" to "true"
end mouseUp
```

De namen van alle mogelijke objecteigenschappen vind je in de *Dictionary*.

Extra-info 5: Werken met datum-gegevens

Elke computer heeft een ingebouwde klok. Het batterijtje zorgt dat die blijft doorlopen als de computer is uitgeschakeld. Via een internetverbinding kun je de klok zelfs automatisch van tijd tot tijd laten gelijkzetten. Je kunt tijd en/of datum vanuit een script laten opvragen met de ingebouwde time- en date-functies:

```
put the date into field "Datum"
```

Je zult dan vaststellen dat de datum wordt weergegeven als '1/21/10'. Als je liever een andere vorm ziet, kun je hem omzetten met de convert-instructie:

```
convert field "datum" to long date
```

Het resultaat zou er voor '11/21/10' zo uitzien: Sunday, January 21, 2010.

Andere vormen (die je eveneens met de convert-instructie berekent) voor de datumweergave zijn:

- seconds: 1264287600;
- dateItems: 2010,1,24,0,0,0,1.

Die zijn belangrijk als je met een datum-gegeven verdere bewerkingen wilt uitvoeren. In de revMedia-documentatie kun je bijvoorbeeld terugvinden dat het laatste item in de dateItems-reeks de weekdag aangeeft (waarbij 1 staat voor zondag).

*Een **item** is een chunk van een reeks waarin de elementen van elkaar gescheiden zijn door komma's. Je kunt de inhoud van zo'n item opvragen met een instructie als:*

```
put item 7 of datum into field "weekdag"
```

5 Procedures en functies

Bij de relatief korte oefeningen die we tot nu toe hebben gemaakt, lukt het meestal wel om in één oogopslag te zien wat het programma doet. Maar bij echt lange programma's is dat niet meer het geval. Bovendien zijn dergelijke 'echte' programma's vaak het werk van meerdere personen, die het programmeerwerk onder elkaar verdelen.

Daarvoor moet je het hoofdprogramma eerst onderverdelen in een aantal deelopdrachten. We nemen als voorbeeld het script dat we maakten voor het berekenen van de grootste gemene deler van twee gehele getallen. Het zou mooi zijn als we dat als volgt zouden kunnen formuleren:

```
on mouseUp
  put empty into fld "deler"
  controleer fld "getal1"
  controleer fld "getal2"
  put ggd(fld "getal1",fld "getal2") into fld "deler"
end mouseUp
```

Als je dit script laat uitvoeren, vertelt revMedia je dat 'controleer' en 'ggd' niet tot de revTalk-terminologie behoren. Om bovenstaand script toch te doen werken moet je op voorhand aan het script uitleggen wat je bedoelt met 'controleer' en 'ggd'. Je moet dat doen in aparte verwerkers, die worden 'aangeropen' vanuit de mouseUp-verwerker. Of anders gesteld: als revMedia de instructie 'controleer' ontmoet, stuurt het een bericht (zie Extra-info 2) dat op zoek gaat naar een verwerker 'on controleer'. Als die op het zoekpad wordt gevonden, wordt hij uitgevoerd en gaat revMedia daarna gewoon door met de rest van de mouseUp-verwerker. Je kunt de controleer-verwerker in hetzelfde script als dat van de mouseUp-verwerker plaatsen, maar het kan ook verder 'stroomafwaarts' op het zoekpad.

'Controleer' *is* een instructie, terwijl 'ggd' *deel uitmaakt* van een instructie. In het eerste geval spreken we van een 'procedure', in het tweede geval van een functie.

Procedure

- een zelfstandige instructie;
- uitgewerkt in een verwerker die begint met 'on';

Functie

- een expressie die deel uitmaakt van een instructie;
- uitgewerkt in een verwerker die begint met 'function';
- naam altijd gevolgd door (), waartussen het argument (kan ook leeg zijn of meerdere argumenten omvatten).

Om vlot met procedures en functies te kunnen werken moet je vertrouwd zijn met nog enkele nieuwe begrippen:

Globale variabelen

Variabelen zijn geheugenplaatsen die hun naam en inhoud verliezen wanneer de verwerker (handler) waarin ze worden gebruikt, is uitgevoerd. Als je in een procedure of functie dezelfde variabelen wilt kunnen gebruiken als in het hoofdprogramma, moeten naam en inhoud behouden blijven. Dat kan door ze als globale variabelen te 'declareren':

```
global getal1, getal2
```

Je plaatst deze declaratie bovenaan in het script (vóór de eerste verwerker). Ze geldt dan voor alle verwerkers in dat script. Je mag ze ook in een verwerker opnemen, maar dan geldt ze alleen voor die ene verwerker. Als je ze in een script met bijvoorbeeld drie verwerkers tussen de eerste en de tweede verwerker plaatst, geldt ze niet voor de eerste maar wel voor de tweede en de derde verwerker.

Parameters

Globale variabelen gebruiken heeft ook zijn nadelen:

- Ze blijven bestaan en dus geheugenruimte innemen zolang je revMedia niet verlaat.
- In procedures en functies moet je dezelfde (globale) variabelenaam gebruiken. Dat maakt het verdelen van het werk, of het gebruik van door anderen geschreven procedures, erg moeilijk.

Die problemen kun je vermijden door **parameters** te gebruiken. Parameters fungeren als een soort doorgeefluik: je duidt op voorhand aan met welke grootheden een procedure of functie werkt, maar ze krijgen maar een inhoud op het ogenblik dat de procedure of functie wordt aangeroepen. De namen die je ze geeft, spelen geen rol. Alleen de **volgorde** is belangrijk.

Een voorbeeld om dat te illustreren — een eenvoudig script dat de som maakt van twee getallen (opgeslagen in twee velden):

```
on mouseUp
  put fld "getal1" into getal1
  put fld "getal2" into getal2
  maak_som getal1, getal2
end mouseUp

on maak_som x1, x2
  put (x1+x2) into fld "som"
end maak_som
```

Bij functies zijn de argumenten de over te dragen parameters:

```
on mouseUp
  put fld "getal1" into getal1
  put fld "getal2" into getal2
  put som(getal1, getal2) into fld "som"
end mouseUp

function som x1, x2
  put (x1+x2) into som
  return som
end som
```

Een functie is een expressie en daarom is de return-instructie belangrijk: zij geeft aan wat de uitkomst van de expressie is. Het hoofdprogramma werkt, na het aanroepen van de functie, verder met die uitkomst.

Opdracht

- 5.1. Herwerk het script van oefening 5, zodat de mouseUp-verwerker er als volgt uitziet:

```
on mouseUp
  global getal1, getal2
  put fld "getal1" into getal1
  put fld "getal2" into getal2
  controleer
  bereken_ggd
end mouseUp
```

- 5.2. Herwerk het nogmaals, zodat de mouseUp-verwerker er nu zo uitziet:

```
on mouseUp
  controleer fld "getal1"
  controleer fld "getal2"
  put fld "getal1" into getal1
  put fld "getal2" into getal2
  bereken_ggd getal1, getal2
end mouseUp
```

- 5.3 Herwerk het nogmaals, zodat de mouseUp-verwerker er nu zo uitziet:

```
on mouseUp
  controleer fld "getal1"
  controleer fld "getal2"
  put fld "getal1" into getal1
  put fld "getal2" into getal2
  put ggd(getal1, getal2) into fld "deler"
end mouseUp
```

De controleer-procedure in de bovenstaande scripts moet nagaan of de ingetikte getallen:

- *niet meer dan 9 cijfers bevatten;*
- *gehele, positieve getallen zijn.*

Als dat niet zo is, verschijnt er een foutmelding .

Extra-info 6: Afolmenu's

In alle programma's geef je instructies via afrolmenu's. Op verschillende computersystemen werken menu's op nagenoeg dezelfde manier, maar toch zijn er kleine verschillen tussen bijvoorbeeld een Macintosh, een Windows-pc of een Unix-systeem.

In revMedia kun je een programma een eigen menu geven. Maar omdat revMedia platformoverschrijdend is en afrolmenu's in de verschillende systemen op telkens andere manieren zijn geïmplementeerd, hebben de ontwikkelaars ervan menu's niet als een apart object-type ingebouwd. In revMedia is een afrolmenu een knop met speciale eigenschappen.

Een knop met die eigenschappen is al voorzien in het objectenpalet. (Je kunt ook onder Objects/New Control 'Pull-down Menu' nemen.) Als je dergelijke knoppen links bovenaan in het venster van je stack plaatst en ze groepeerd, functioneren ze als een afrolmenu. Een aanverwant knoptype is overigens het popup-menu, dat nagenoeg op dezelfde manier functioneert, maar dat je doorgaans niet bovenaan in een venster plaatst.

Zowel pulldown als popup reageren op een menuPick-message en je kunt dan ook een verwerker maken zoals:

```
on menuPick keuze
  if keuze is "Keuze 1" then
    put "A" into fld "antwoord"
  else if keuze is "Keuze 2" then
    put "B" into fld "antwoord"
  else if keuze is "Keuze 3" then
    put "C" into fld "antwoord"
  end if
end menuPick
```

of korter, als je alleen maar de gekozen waarde in een veld wilt plaatsen:

```
on menuPick keuze
  put keuze into fld "antwoord"
end menuPick
```

Je kunt in plaats van 'if...else if' ook gebruik maken van switch- en case-structuren, waarover je meer informatie vindt in de *Dictionary*. Met revMedia is overigens ook een *Menu Builder* meegeleverd, die het maken van afrolmenu's een stuk vergemakkelijkt. Maar eigenlijk heeft dat maar zin als je echt *stand-alone* applicaties wilt ontwikkelen. Voor de eenvoudige toepassingen die we in deze cursus leren bouwen, is een menubalk met een volledige set afrolmenu's eigenlijk niet nodig.

6 Bestanden lezen en schrijven

6.1 ASCII-bestanden lezen

RevMedia kan gegevens die zijn opgeslagen in een ASCII/TEXT-bestand zonder veel moeite importeren. In een tekstverwerkingsprogramma gaat zo iets meestal nog makkelijker: daar volstaat het een dergelijk bestand te openen (het commando daarvoor bevindt zich in het "File"-menu). Bij revMedia is dat niet mogelijk, omdat je ook moet kunnen opgeven in welke container(s) de gegevens terecht moeten komen. De instructies daartoe neem je op in een script.

Vooraleer je echt gegevens uit een TEXT-bestand kunt lezen moet je het bestand openen met de instructie

```
open file <filename>
```

waarin <filename> de naam van het te openen bestand is. Als dat bestand zich niet dezelfde map bevindt als de stack, moet je de volledige toegangsnaam of 'pathname' van het bestand opgeven. Zo'n toegangsnaam ziet er als volgt uit:

Harde Schijf/Bestanden/Oefeningen/Tekst.txt.

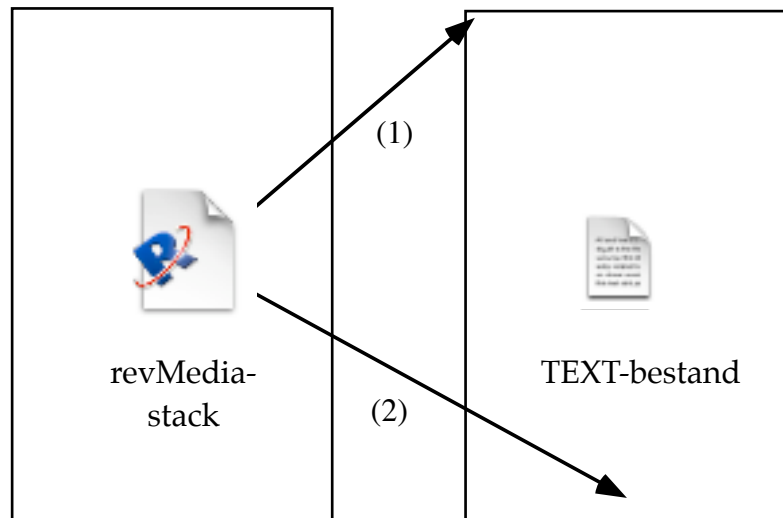
'Tekst.txt' is de eigenlijke bestandsnaam. Die bevindt zich in een map met naam 'Oefeningen', die op haar beurt in de map 'Bestanden' steekt. En die staat op een schijf ('volume') die 'Harde Schijf' heet. Zoals we in het voorbeeld verder in dit hoofdstuk zien, kun je overigens revMedia zelf laten uitzoeken waar een bepaald bestand zich bevindt met de instructie

```
answer file "Kies een bestand:"
```



Die brengt een dialoogvenster op het scherm zoals je altijd krijgt bij het openen van een bestand in een willekeurig programma en plaatst de gekozen

bestandsnaam in de variabele 'it'. Uiteraard mag je i.p.v. "Kies een bestand:" een andere zelf gekozen zin invullen. Je mag de instructie zelfs aanvullen met "... of type <bestandstype>". Als je bijvoorbeeld 'TEXT' opgeeft, zullen in dit dialoogvenster alleen maar de TEXT- of ASCII-bestanden te zien zijn.



De open-instructie brengt de inhoud van het bestand niet over naar (een veld of variabele in) een revMedia-stack (in het RAM-geheugen van de computer), maar legt alleen maar een 'link' tussen beide. Je zou je kunnen voorstellen dat er vanuit de stapel een verbindingskanaal — programmeurs spreken van een wijzer ('pointer') — wordt geplaatst naar het begin van het TEXT-bestand: (1) in de afbeelding hierboven. Het bestand is dan klaar voor import of export van tekst. Als er geen bestand bestaat met de opgegeven toegangsnaam, wordt het op dat ogenblik gecreëerd.

De eigenlijke leesopdracht geef je met de instructie

```
read from file <filename> until <character>
```

waarin <filename> de toegangsnaam is en <character> het karakter tot waar de leesopdracht moet lopen. Een andere toegelaten formulering is

```
read from file <filename> for <aantal>
```

waarin <aantal> het aantal karakters is dat moet worden ingelezen. De wijzer wordt dan naar de met 'until' of 'for' aangegeven plaats in het TEXT-bestand verplaatst: (2) in de afbeelding hierboven.

De ingelezen tekst — alles tussen de posities (1) en (2) van de wijzer — wordt in de variabele 'it' geplaatst — dezelfde 'it' die ook automatisch wordt gecreëerd in de get- en answer-instructie. Met een toekenningsopdracht (put) kun je de inhoud ervan naar een container overbrengen:

```
put it into card field "Tekst"
```

Als je op het einde van het TEXT-bestand komt — dus als de wijzer naar lege bytes wijst —, krijgt 'it' de waarde 'empty'. Maar maak je geen zorgen: de put-instructie brengt alle bytes die wél gegevens bevatten, toch netjes over.

Je kunt van deze empty-waarde wel gebruik maken om de lees-opdracht af te breken als alle tekst is ingelezen, of — zoals we in een voorbeeld verder zien — te controleren of alle tekst inderdaad is ingelezen:

```
if it is empty then ...
```

Nadat je alle gewenste informatie naar de de revMedia-stack hebt overgebracht, dient het TEXT-bestand gesloten te worden. De verbinding tussen de stapel en het bestand wordt daardoor verbroken. Dat doe je met de instructie

```
close file <filename>
```

Een voorbeeldscript:

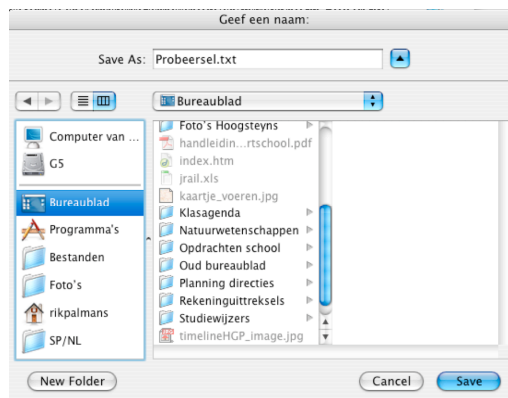
```
on mouseUp
  answer file "Kies een bestand:"
  if it is empty then
    exit mouseUp
  else
    put it into naam
  end if
  open file naam
  read from file naam until EOF
  put it into field "Tekst"
  close file naam
end mouseUp
```

6.2 ASCII-bestanden schrijven

Het omgekeerde van de read-opdracht (om gegevens in te lezen) is de write-instructie, waardoor je informatie uit een revMedia-stack kunt exporteren naar een TEXT- of ASCII-bestand. De werkwijze is bijna volledig gelijk: eerst open je een bestand, daarna schrijf je gegevens weg en tenslotte sluit je het bestand.

De inhoud van een veld zou je dus als volgt kunnen wegschrijven:

```
on mouseUp
  ask file "Geef een naam:" with "Probeersel"
  put it into bestandsnaam
  open file bestandsnaam
  write field "Tekst" to file bestandsnaam
  close file bestandsnaam
end mouseUp
```



Het tegenovergestelde van de 'answer file'-instructie in het vorige script is hier 'ask file', dat een dialoogvenster op het scherm brengt zoals bij de opdracht "Bewaar (als)" in andere programma's. De gebruiker kan ook hier nog altijd de naam van het bestand wijzigen, of de map waarin hij de tekst geplaatst wil zien. De opgegeven toegangsnaam wordt in de variabele 'it' gestopt, en kan daarna naar een andere variabele worden overgebracht.

Extra-info 7: ASCII- of TEXT-bestanden

Hoe onthoudt een computer eigenlijk een tekst in zijn intern geheugen? Eenvoudig voorgesteld bestaat dat uit miljoenen minuscule schakelaartjes die op ‘aan’ of op ‘uit’ kunnen staan. ‘Aan’ kun je voorstellen met het cijfer 1, ‘uit’ met 0. Elke 1 of 0 is een ‘bit’ (= binary digit). Met alleen de cijfers 1 en 0 kun je perfect een volledig getallenstelsel opbouwen, waarin alle wiskundige bewerkingen mogelijk zijn: het binaire of tweetallige getallenstelsel. Hoe dat precies gaat, heeft voor ons verder geen belang, maar weten dat het bestaat kan nooit kwaad. Ook op een harde schijf staat alles genoteerd in de vorm van bits: een vlekje dat wél gemagnetiseerd is staat voor een 1, en een niet-gemagnetiseerd voor een 0. Idem op een cd of dvd: daar worden eentjes en nulletjes voorgesteld door putjes van verschillende diepte (die al dan niet het licht van een laserstraal weerkaatsen).

Verder moet je nog weten dat computers met reeksen van bits werken — meestal met een lengte van 8 bits, wat we een ‘byte’ noemen. Zo neemt elk ingetikt karakter van een tekst één byte in beslag.

Een byte lezen als een binair getal is voor een computer een koud kunstje, maar een mens heeft heel wat moeite met al die nulletjes en eentjes. Je kunt dat oplossen door het getal van het binaire getallenstelsel om te zetten in het decimale — dat we in het dagelijks leven gewend zijn te gebruiken — of in het hexadecimale, waar computerprogrammeurs meestal wel vertrouwd mee zijn. Hoe je een getal precies omrekent van het ene naar het andere getallenstelsel, hoef je niet te weten om de rest van dit verhaal te kunnen volgen. Je vindt er meer informatie over in algemene handboeken over informatica. We geven hieronder alleen even een voorbeeld.

Zo wordt de hoofdletter A door de volgende byte voorgesteld:

[0]1000001.

Waarom we de eerste bit tussen haakjes plaatsen, leggen we dadelijk uit.

Dat komt overeen met het decimale getal 65, of met het hexadecimale 41.

In de tabellen op de vorige en deze pagina zie je met welke decimale, respectievelijk hexadecimale waarden alle andere karakters overeenkomen. Deze waarden zijn bij afspraak vastgelegd in de zg. *American Standard Code for Information Interchange* (ASCII). Eigenlijk is het alleen maar voor de karakters

tot en met 127 een echte standaard. Vanaf 128 is dat niet meer het geval en hanteert het Macintosh-systeem bijvoorbeeld andere codes dan Windows. Bij uitwisseling van teksten tussen de twee loopt het dus prima voor de letters met een code lager dan 128, maar is het resultaat voor de hogere waarden een puinhoop.

Het probleem is de laatste jaren door de opkomst van het internet nog een stuk acuter geworden. Het internet is immers bij uitstek een platformoverschrijdend medium. Volkomen eensgezindheid is er nog lang niet. Er zijn verschillende standaarden vastgelegd door de International Organization for Standards. Die worden (bijvoorbeeld in webpagina's) aangeduid met codes als 'ISO-8859-1'. ISO-standaarden voldoen nog niet helemaal; zo komt bijvoorbeeld het €-teken er niet in voor. Microsoft heeft daarom een eigen 'charset' gedefinieerd, Windows-1252. Intussen breekt stilaan een systeem door, Unicode, dat uiteindelijk alle lettertekens die in welk alfabet dan ook worden gebruikt, moet omvatten.

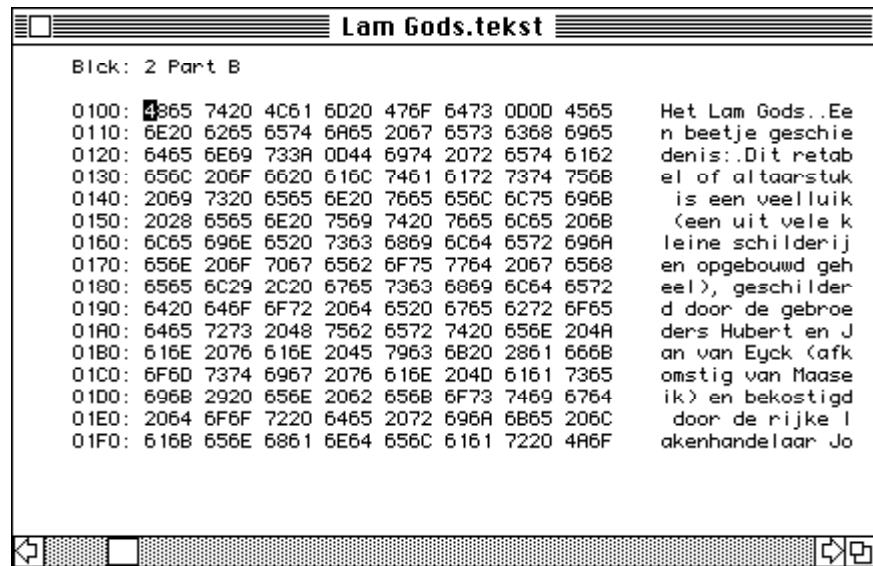
En hoe zit het dan met die eerste bit? Wel, de ASCII-code is oorspronkelijk ontworpen om gegevens tussen computers via onder meer telefoonlijnen uit te wisselen. Voor de overdracht zelf werden maar zeven van de acht bits gebruikt. De overblijvende bit was de 'pariteitsbit', die zodanig gekozen werd dat het aantal eentjes in de volledige byte even is en dus een controle vormde op een correcte overdracht van de gegevens. Dat verklaart meteen waarom de hoogste ASCII-waarde oorspronkelijk 127 bedroeg: het aantal getallen (0 inbegrepen) bedraagt dan 128, wat gelijk is aan 2 tot de 7de macht. Door ook de pariteitsbit te gebruiken voor informatie-opslag of -overdracht, krijg je 256 (= 2 tot de 8ste macht) verschillende codes ter beschikking.

	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
0		□		0	@	P	`	p	Á	ê	†	∞	¿	-	‡	☛
1	□	□	!	1	A	Q	a	q	Ã	ë	°	±	¡	—	·	Ö
2	□	□	"	2	B	R	b	r	Ç	í	¢	≤	¬	"	,	Û
3	□	□	#	3	C	S	c	s	É	ì	£	≥	√	"	„	Ü
4	□	□	\$	4	D	T	d	t	Ë	ï	§	¥	f	'	%	Ù
5	□	□	%	5	E	U	e	u	Ö	ï	•	μ	≈	'	Â	ı
6	□	□	&	6	F	V	f	v	Ü	ñ	¶	ð	Δ	÷	Ê	ˆ
7	□	□	'	7	G	W	g	w	á	ó	ß	Σ	«	♦	Á	˜
8	□	□	(8	H	X	h	x	à	ò	©	Π	»	ÿ	Ë	˘
9		□)	9	I	Y	i	y	â	ô	©	π	...	ÿ	Ë	˘
10	□	□	*	:	J	Z	j	z	ä	ö	™	∫		ı	ı	˙
11	□	□	+	;	K	[k	{	ã	õ	'	ª	À	α	ı	˙
12	□	□	,	<	L	l	l		â	ú	"	º	Ã	<	ı	˙
13		□	-	=	M]	m	}	ç	ù	≠	Ω	Ö	>	ı	˙
14	□	□	.	>	N	^	n	~	é	û	Æ	æ	œ	fi	Ö	˘
15	□	□	/	?	O	_	o	□	è	ü	Ø	ø	œ	fi	Ö	˘

	0	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0		□		0	@	P	`	p	Á	ê	†	∞	¿	-	‡	☛
1	□	□	!	1	A	Q	a	q	Ã	ë	°	±	¡	—	·	Ö
2	□	□	"	2	B	R	b	r	Ç	í	¢	≤	¬	"	,	Û
3	□	□	#	3	C	S	c	s	É	ì	£	≥	√	"	„	Ü
4	□	□	\$	4	D	T	d	t	Ë	ï	§	¥	f	'	%	Ù
5	□	□	%	5	E	U	e	u	Ö	ï	•	μ	≈	'	Â	ı
6	□	□	&	6	F	V	f	v	Ü	ñ	¶	ð	Δ	÷	Ê	ˆ
7	□	□	'	7	G	W	g	w	á	ó	ß	Σ	«	♦	Á	˜
8	□	□	(8	H	X	h	x	à	ò	©	Π	»	ÿ	Ë	˘
9		□)	9	I	Y	i	y	â	ô	©	π	...	ÿ	Ë	˘
A	□	□	*	:	J	Z	j	z	ä	ö	™	∫		ı	ı	˙
B	□	□	+	;	K	[k	{	ã	õ	'	ª	À	α	ı	˙
C	□	□	,	<	L	l	l		â	ú	"	º	Ã	<	ı	˙
D		□	-	=	M]	m	}	ç	ù	≠	Ω	Ö	>	ı	˙
E	□	□	.	>	N	^	n	~	é	û	Æ	æ	œ	fi	Ö	˘
F	□	□	/	?	O	_	o	□	è	ü	Ø	ø	œ	fi	Ö	˘

De bovenste figuur geeft een overzicht van het alfabet, aangevuld met diakritische en andere speciale tekens, met hun bijbehorende ASCII-codes in het decimale getallenstelsel. Programmeurs werken echter op bytes-niveau veel vaker met het hexadecimale stelsel. Dat omvat nog zes extra-cijfers (die we voorstellen met de letters A tot en met F). Het voordeel ervan is dat je dan elke byte (8 bits) kunt voorstellen met een uit twee cijfers bestaand getal.

Er bestaan diverse programma's waarmee je de inhoud van een bestand op schijf kunt opvragen. Dit is een voorbeeld van wat je dan op het scherm krijgt:



```

Lam Gods.tekst
Blck: 2 Part B
0100: 4865 7420 4C61 6D20 476F 6473 0000 4565   Het Lam Gods..Ee
0110: 6E20 6265 6574 6A65 2067 6573 6368 6965   n beetje geschie
0120: 6465 6E69 733A 0044 6974 2072 6574 6162   denis:.Dit retab
0130: 656C 206F 6620 616C 7461 6172 7374 756B   el of altaarstuk
0140: 2069 7320 6565 6E20 7665 656C 6C75 696B   is een veelluik
0150: 2028 6565 6E20 7569 7420 7665 6C65 206B   (een uit vele k
0160: 6C65 696E 6520 7363 6869 6C64 6572 696A   leine schilderij
0170: 656E 206F 7067 6562 6F75 7764 2067 6568   en opgebouwd geh
0180: 6565 6C29 2C20 6765 7363 6869 6C64 6572   eel), geschilder
0190: 6420 646F 6F72 2064 6520 6765 6272 6F65   d door de gebroe
01A0: 6465 7273 2048 7562 6572 7420 656E 204A   ders Hubert en J
01B0: 616E 2076 616E 2045 7963 6820 2861 666B   an van Eyck (afk
01C0: 6F6D 7374 6967 2076 616E 204D 6161 7365   omstig van Maase
01D0: 6968 2920 656E 2062 6568 6F73 7469 6764   ik) en bekostigd
01E0: 2064 6F6F 7220 6465 2072 696A 6865 206C   door de rijke l
01F0: 616B 656E 6861 6E64 656C 6161 7220 4A6F   akenhandelaar Jo
```

Rechts staat de tekst zelf. Bovenaan en in de kolom links staan kengetallen om deze tekst in het bestand te situeren, en in het midden staan de hexadecimale getalwaarden van alle karakters. Daarom heet een dergelijke weergave van de bestandsinhoud wel eens een 'hexdump'.

In een 'gewoon' bestand, zoals je dat maakt met een willekeurig toepassingsprogramma, zul je naast de eigenlijke data ook een hoop andere codes terugvinden. Die bepalen bijvoorbeeld de vormgevingsaspecten en op dat vlak is er totaal geen eenvormigheid tussen diverse programma's. Om uitwisseling mogelijk te maken, zou je die codes er allemaal moeten uithalen. Dat is nu precies wat er gebeurt als je een bestand bewaart in ASCII- of TEXT-vorm: alleen de tekstgegevens worden opgeslagen, alle andere worden gewoon weggelaten. Je verliest dus wel een deel van de informatie (vormgeving, grafische elementen), maar het voordeel is dat dit tekstbestand door elk willekeurig programma kan worden geopend.

Naarmate computers een steeds grotere diversiteit aan gegevens moesten kunnen uitwisselen — niet alleen meer teksten, maar ook meetwaarden, afbeeldingen, ... — zijn er nog andere standaarden ontwikkeld.

Dat is vooral belangrijk geworden nu via het internet computers die onder verschillende besturingssystemen draaien met elkaar verbonden zijn.

HTML, de standaard waarin webpagina's zijn opgemaakt, is nog te be-

schouwen als een afgeleide van het ASCII-systeem. De HTML-code zelf is dus zelfs met een eenvoudige tekstverwerker (die alleen maar TEXT herkent) te lezen, maar voor de correcte weergave van de inhoud is misschien ondersteuning nodig — zeker als het gaat om tekst in een ander alfabet dat het onze. Hoe een browser in staat is om code en inhoud tot een nieuw geheel — de webpagina — om te vormen, is het onderwerp van een volgende cursus.