Module 7 Expressies, procedures en functies

Onderwerp	Definiëren van wiskundefuncties als Maple-procedures.
Voorkennis	geen.
Expressies	`unapply, ->, proc`
Zie ook	Module 1, 3, 6, 25, 28.

We hebben al diverse voorbeelden gezien van procedureaanroepen: `simplify`, `solve` en `plot` zijn procedures en `simplify(sqrt(1))` en `plot(sin(t),t=-Pi..Pi)` zijn procedure*aanroepen*. Ook wiskundige functies worden door Maple opgevat als procedures. Het begrip procedure zullen we hier in het kort bespreken. Voor meer details verwijzen we naar Module 28.

7.1 Procedures

Procedures bestaan in feite uit twee delen: een zogenaamde *heading* en een *body*.
Grof gezegd is de heading datgene wat we nodig hebben bij de aanroep. Dus als we Maple het commando geven `diff(2*x,x)`, dan gebruiken we alleen de heading van de procedure `diff`, namelijk

```
diff(arg1,arg2)
```

De body van de procedure is in de meeste gevallen voor ons onzichtbaar. Hier vindt de feitelijke uitvoering plaats van het werk dat de procedure moet doen. De body staat in contact met de rest van de Maple-sessie via de *parameters* of *argumenten* die in de aanroep (in de heading) worden meegegeven. De body is een lijst Maple-instructies, die aangeven wat er gedaan moet worden met de argumenten die meegegeven zijn bij de aanroep van die procedure.

`->`

Eén keer hebben we al wel te maken gehad met zichtbare bodies, namelijk in het geval van de definitie van een functie met het pijltje (`->`). Bijvoorbeeld, voor Maple definieert `f:=x->2*x` een procedure met heading `f(x)` en body `2*x`. Uiteraard is `x` in de definitie `f:=x->2*x` slechts een dummy-variabele. We noemen deze dummy ook wel de *formele* parameter van de procedure `f`, dus de parameter die gebruikt is bij de definitie van `f`. Wanneer we bijvoorbeeld `f(3)` opvragen, dan is de parameter `3` nu geen dummy. We noemen `3` wel

de *actuele* parameter voor de procedure `f`, dus de parameter waarop de procedure `f` moet worden toegepast.
In de body staat in dit geval dat het argument (de actuele parameter) waarmee `f` wordt aangeroepen met 2 moet worden vermenigvuldigd. De afwerking van de Maple-procedure gaat in drie stappen. We nemen als voorbeeld de aanroep

```
y:=1:  f(3*y);
```

met `f` de procedure `x -> 2*x`.

(1) Eerst wordt gekeken wat de actuele parameter is (in dit geval $3y$). Deze wordt zover mogelijk uitgerekend tot de waarde 3×1.
(2) Vervolgens wordt elke formele parameter in de body vervangen door de berekende waarde van de actuele parameter (dus `2*x` wordt vervangen door `2*3`).
(3) Dan pas wordt de opdracht in de body uitgevoerd: `2*3` wordt 6, en dit resultaat wordt getoond.

In grote lijnen gaat deze afhandeling op voor alle Maple-procedures, afgezien van nog wat complicaties die in Module 25 en 28 aan de orde komen.

7.2 Procedures en expressies

Procedures zullen uitgebreider aan bod komen in Module 28. We gaan hier nog in op het verschil tussen procedures en expressies. Dit verschil bespreken we aan de hand van het voorbeeld uit §6.3, het berekenen van de afgeleide van een functie.
Eerst keren we terug naar onze functie `f := x -> 2*x`. Hier is `f` de naam van een Maple-procedure. Echter, wanneer we `f` toepassen op een argument `y`, dan is `f(y)` natuurlijk geen Maple-*procedure* meer, maar de *expressie* `2*y`. Het is belangrijk een goed onderscheid te maken tussen de *naam* van de procedure (namelijk `f`) en het resultaat van een toepassing (*application*) van die procedure (namelijk `f(y)`).

Nu beschouwen we de afgeleide van `f`. Wanneer f een functie is, dan is f' wiskundig gezien ook een functie. Stel dat f' met Maple berekend wordt door het commando `df := x -> diff(f(x),x)`, zoals we in §6.3 hebben gezien. Dit levert echter niet een Maple-procedure op waarmee we zomaar `df(3)` kunnen uitrekenen. We kunnen nu, na wat er in §7.1 over de afwerking van een procedure-aanroep is vermeld, begrijpen hoe dit komt. We beschouwen weer de opdracht

```
df := x -> diff(f(x),x);
```

Dit wordt door Maple opgevat als procedure met heading `df(x)` en body `diff(f(x),x)`. Wanneer we nu `df(3)` opvragen, dan wordt de expressie `3` overal waar `x` als formele parameter in de body voorkomt voor `x` gesubstitueerd. De aanroep `df(3)` levert dus voor de body de uitdrukking: `diff(f(3),3)` en dit mag niet, want je mag alleen differentiëren naar een variabele.

Ook `df := diff(f(x),x)` resulteert niet in een geldig functievoorschrift voor f'. Dit komt omdat het resultaat van de uitdrukking `diff(f(x),x)` geen procedure is. Eigenlijk zou je in dit geval willen dat je op een of andere manier aan Maple duidelijk kunt maken dat de expressie `df` moet worden opgevat als een functie met `x` als

`unapply` variabele. Dit kan met het commando `unapply`:

```
df := unapply(diff(f(x),x),x);
```

Hiermee is `df` een Maple-procedure geworden met de formele parameter `x`.
Het commando

```
unapply(expressie,x);
```

maakt dus van een uitdrukking `expressie` waarin de naam `x` voorkomt een procedure met formele parameter `x` en body gegeven door `expressie`.

Functies van meer variabelen gaan net zo: als `expressie` een expressie is waarin de namen `x` en `y` voorkomen, dan maakt

```
f := unapply(expressie,x,y);
```

van `f` de functie $(x, y) \mapsto expressie$.

Voorbeeldopgave

Definieer de functie van x gegeven door $f : x \mapsto \int_0^x e^{2t}dt$.

Voorbeeldsessie

```
> k := int(exp(2*t), t=0..x);
```

$$k := -\frac{1}{2} + \frac{1}{2}\, e^{(2\,x)}$$

```
> f := unapply(k,x);
```

$$f := x \to -\frac{1}{2} + \frac{1}{2}\, e^{(2\,x)}$$

```
> f(2);
```

$$-\frac{1}{2} + \frac{1}{2}\, e^{4}$$

We proberen het op een andere manier:

```
> g := x -> k;
```

$$g := x \to k$$

```
> g(2);
```

$$-\frac{1}{2} + \frac{1}{2}\, e^{(2\,x)}$$

Dat gaat dus niet goed!

Toelichting

Met `k := int(exp(2*t),t=0..x)` wordt `k` een expressie waarin `x` voorkomt, en `unapply` zorgt ervoor dat dit wordt opgevat als een functie van `x`.
Men zou verwachten dat `g` hetzelfde is als `f`. Er valt direct een verschil op. Op het statement `f := unapply(k,x)` reageert Maple anders dan op het statement `g := x -> k`. In het eerste geval is de `k` vervangen door de eerder uitgerekende integraal, in het tweede geval laat Maple de `k` gewoon staan. Bovendien levert toepassen van de procedure `g` op het argument `2` niet het gewenste resultaat. ◇

In de volgende paragraaf zullen we de werking van `->` en `unapply` aan een nadere analyse onderwerpen. In §7.4 komt nog een derde manier om functies te definiëren aan de orde.

7.3  Verschil tussen `->` en `unapply`

We bekijken nogmaals de voorbeeldsessie van de vorige paragraaf, maar we geven nu de expressie `k` een andere waarde nadat `f` en `g` gedefinieerd zijn:

Voorbeeldsessie

```
> k := int(exp(2*t), t=0..x):
  f := unapply(k,x);
```

$$f := x \to -\frac{1}{2} + \frac{1}{2}\, e^{(2\,x)}$$

```
> g := x -> k;
```

$$g := x \to k$$

We maken van k een andere expressie en we kijken wat er met f en g gebeurt:

```
> k := a*x^3:
  f(y); g(y);
```

$$-\frac{1}{2} + \frac{1}{2}\, e^{(2\,y)}$$

$$a\,x^3$$

Toelichting

De procedure `g` is gedefinieerd als `x -> k`. Als we `g` toepassen op het argument `2`, dan wordt éérst in `k` elke `x` die daarin voorkomt vervangen door `2`. In de expressie `k` (die in dit stadium alleen uit de *naam* "`k`" bestaat) komt `x` niet voor, dus er gebeurt nog niets. Ten slotte wordt deze `k` 'uitgerekend'; dit blijkt volgens het vierde statement $a\,x^3$ te zijn, en dit kan niet verder worden uitgerekend omdat x geen waarde heeft.
In de definitie van `f` als `unapply(k,x)` is `unapply` zélf een procedure. Dat betekent dat alles wat tussen haakjes staat éérst wordt geëvalueerd. In dit geval wordt dus `k` vervangen door de waarde die `k` op dat moment heeft, namelijk $\frac{1}{2}e^{2x} - \frac{1}{2}$. Vervolgens zorgt `unapply` ervoor dat deze expressie als functie van x wordt gedefinieerd. Bij de toepassing (*application*) van `f` op het argument `2` wordt in de expressie $\frac{1}{2}e^{2x} - \frac{1}{2}$ de x vervangen door 2 (en het resultaat zo mogelijk verder geëvalueerd), hetgeen tot het beoogde resultaat leidt. ◇

Samengevat is het verschil tussen `->` en `unapply` dus:

- `x ->` ⟨*expressie*⟩`:` levert een procedure waarin ⟨*expressie*⟩ in *letterlijke,* on*geëvalueerde* vorm als body optreedt;
- `unapply(`⟨*expressie*⟩`,x):` levert een procedure waarin de uitdrukking ⟨*expressie*⟩ in *volledig geëvalueerde* vorm als body optreedt.

We herhalen nu nog eens het voorbeeld van §6.3 in een iets gewijzigde vorm:

Voorbeeldopgave

Gegeven de functie $f(x) = e^{2x}$.
Definieer de functies $h(x) = \int_0^x f(t)\,dt$ en $i(x) = f'(x)$.

Voorbeeldsessie

```
> h := x -> int(exp(2*t), t=0..x);
```

$$h := x \rightarrow \int_0^x e^{(2\,t)}\,dt$$

```
> h(y);
```

$$-\frac{1}{2} + \frac{1}{2}\,e^{(2\,y)}$$

```
> h(2);
```

$$-\frac{1}{2} + \frac{1}{2}\,e^{4}$$

```
> i := x -> diff(exp(2*x),x);
```

$$i := x \rightarrow \mathrm{diff}(e^{(2\,x)},\, x)$$

```
> i(y);
```

$$2\,e^{(2\,y)}$$

```
> i(2);
```

```
Error, (in i) invalid input: diff received 2, which is not
valid for its 2nd argument
```

```
> j := unapply( diff(exp(2*x),x), x);
```

$$j := x \rightarrow 2\,e^{(2\,x)}$$

```
> j(x), j(y), j(2);
```

$$2\,e^{(2\,x)},\ 2\,e^{(2\,y)},\ 2\,e^{4}$$

Toelichting

De definitie van `h` in deze voorbeeldsessie levert dus nog iets anders op dan de definitie van `f` en `g` in de vorige sessie. De aanroep `h(2)` resulteert in eerste instantie in het voorschrift om de `x` te vervangen door `2` in `int(exp(2*t),t=0..x)`, dat geeft $\int_0^2 e^{2t}dt$, en dit wordt (pas) daarna geëvalueerd tot $\frac{1}{2}e^4 - \frac{1}{2}$. Na de aanroep `f(2)` vindt vervanging van `x` door de waarde `2` direct in de uitdrukking $\frac{1}{2}e^{2x} - \frac{1}{2}$ plaats.
Bij de procedure `i` gaat het net zo als bij de procedure `h`, alleen betekent dat hier dat de uitdrukking $e^{2\times 2}$ naar 2 moet worden gedifferentieerd, en dat resulteert in een foutmelding. Bij de definitie van `j` met `unapply` gaat het goed: e^{2x} wordt éérst gedifferentieerd, en het *resultaat* wordt als functie van x opgevat. ◇

Ten slotte nog een terminologie-kwestie. Het wiskundige begrip *functie* komt dus overeen met het Maple-begrip *procedure*; het wiskundige begrip functie*waarde* is in Maple (het resultaat van) een procedure*aanroep*. Als `f(x)` niet berekend kan worden, bijvoorbeeld omdat `f` ongedefinieerd is zoals in §6.9 gebeurd is, wordt de Maple-expressie `f(x)` ook *function* genoemd, zie verder Module 26. Dus:

> **Wiskunde:** als f een functie, dan is $f(x)$ een functiewaarde;
> **Maple:** als `f` een procedure, dan is `'f(x)'` een function.

7.4 Drie manieren om Maple-procedures te definiëren

Om de functie $f : x \mapsto x^2+1$ in Maple te definiëren hebben we tot nu toe twee verschillende manieren gezien, namelijk `f := x -> x^2+1` en `f := unapply(x^2+1, x)`.
Er is nog een derde manier, namelijk

`proc`

```
f := proc(x) x^2+1 end proc;
```

Hiermee wordt *precies dezelfde* procedure `f` gemaakt als met `unapply`. Uiteraard is `proc` een afkorting van 'procedure'. Bovenstaande definitie kan worden gelezen als: "`f` wordt een procedure met één parameter (`x`) en body (functievoorschrift) $x^2 + 1$." Anders gezegd: `f` verwerkt een input `x` tot output $x^2 + 1$.
Het voordeel van een definitie met `proc` is dat er bij de verwerking van input naar output veel ingewikkelder dingen kunnen gebeuren dan alleen het 'invullen van een formule' zoals bij `->` en `unapply`. In het bijzonder kunnen daarbij *hulpvariabelen* of *lokale variabelen* worden gebruikt, die alléén 'binnen de procedure', dat wil zeggen *tijdens* de verwerking, een waarde kunnen krijgen. Vooruitlopend op een uitgebreide behandeling in Module 28 laten we hier een eenvoudig voorbeeld zien.
In Module 5 hebben we het volgende voorbeeld bekeken (zie blz. 62):

> Gevraagd een functie $p(x)$ van de vorm ax^2+bx+c,
> waarvoor geldt: $p(1) = 1$, $p(2) = 6$ en $p(3) = 2$.

De daarbij gegeven oplossing (zie de voorbeeldsessie) zou gemakkelijk kunnen worden herhaald voor andere waarden van $p(1), p(2)$ en $p(3)$. Maar we kunnen er ook een procedure van maken, laten we zeggen met de naam `pol`. Als we van de gevraagde functie p eisen dat $p(1) = y_1$, $p(2) = y_2$ en $p(3) = y_3$, dan wordt `pol` een procedure die de input y_1, y_2, y_3 verwerkt tot een output van de vorm $x \mapsto \ldots$ Dat gaat als volgt:

Voorbeeldsessie

```
> pol := proc(y1,y2,y3)
    local p, x, a, b, c, stelsel, s;
    p := x -> a*x^2 + b*x + c;
    stelsel := {p(1)=y1, p(2)=y2, p(3)=y3};
    s := solve( stelsel, {a,b,c} );
    subs( s, p(x) );
    unapply( %, x )
  end proc;
```

```
pol := proc(y1, y2, y3)
local p, x, a, b, c, stelsel, s;
    p := x → a * x^2 + b * x + c;
    stelsel := {p(1) = y1, p(2) = y2, p(3) = y3};
    s := solve(stelsel, {a, b, c});
    subs(s, p(x));
    unapply(%, x)
end proc
```

```
> p := pol(1,6,2);
```

$$p := x \to -\frac{9}{2}\,x^2 + \frac{37}{2}\,x - 13$$

```
> q := pol(4,3,2);
```

$$q := x \to 5 - x$$

Toelichting

Vergelijk met de voorbeeldsessie op blz. 62. De statements tussen de regel met `local` en `end proc` zijn precies dezelfde als die we in Module 5 afzonderlijk hebben uitgevoerd. Het enige verschil is dat we het eindresultaat: '`unapply(%, x)`' nu niet aan een variabele hoeven toe te kennen omdat dit automatisch gebeurt bij de procedure*aanroep*. Achter het woord `local` staan de namen van de gebruikte hulpvariabelen.

`local`

Als we na de hele procedure te hebben ingevoerd op [Enter] drukken, dan rekent Maple nog niets uit, maar kijkt alleen of er geen fouten in staan. Als er bijvoorbeeld een puntkomma is vergeten, of als u al op [Enter] drukt voordat `end proc` is ingevoerd, dan reageert het met een foutmelding. Om te laten zien dat er geen fouten zijn gevonden, en Maple, zolang er geen `restart` is gegeven, de procedure `pol` 'kent', toont het de hele procedure nog eens in een nette opmaak.
De gevraagde functie is te verkrijgen door `pol` met de juiste argumenten aan te roepen. Merk op dat er inderdaad een eerstegraadsfunctie uit komt als de punten toevallig op een lijn liggen. ◇

Opgave 7.1

Definieer de functie

$$g : x \mapsto \frac{d}{dx}\sin(\sin(x))$$

en bereken $g(1)$, $g(2)$ en $g(3)$ op de volgende manieren:

(a) Eerste manier: Werk zoveel mogelijk met functies (`->` en/of `unapply`) zodat u de gevraagde functiewaarden $g(1)$, $g(2)$, $g(3)$ krijgt met de Maple-commando's `g(1);` enzovoort.

(b) Tweede manier: Werk uitsluitend met *expressies*; in dit geval zal uw eerste Maple-regel bijvoorbeeld zijn:

```
restart; S := sin(sin(x));
```

Voor het berekenen van de functiewaarden $g(1)$ enzovoort op het laatst zult u dan een `subs`- of `eval`-commando nodig hebben.

Opgave 7.2

Definieer een functie g die een primitieve is van de functie

$$f : x \mapsto x^3 \sin(x)$$

en bereken $g(1)$, $g(2)$ en $g(3)$.

Opgave 7.3

Gegeven

$$f : x \mapsto \frac{1+x}{1-x} \quad \text{en} \quad g : x \mapsto \frac{1-x^2}{\cos(x)}\,.$$

Bereken $(f \circ g)'(x)$.

Opgave 7.4

Maak twee procedures `evendeel` en `onevendeel`, zodat, met f een *functie*:

- als $f_{\mathrm{E}} = \text{evendeel}(f)$ er geldt dat $f_{\mathrm{E}}(x) = \frac{1}{2}\left(f(x) + f(-x)\right)$; en
- als $f_{\mathrm{O}} = \text{onevendeel}(f)$ er geldt dat $f_{\mathrm{O}}(x) = \frac{1}{2}\left(f(x) - f(-x)\right)$.

Gebruik uw procedures om de volgende vragen te beantwoorden:

(a) Toon aan dat f_{E} altijd een even functie is en f_{O} altijd een oneven functie is;

(b) Laat zien dat geldt: $f = f_{\mathrm{E}} + f_{\mathrm{O}}$;

(c) Als $f(x) = e^x$, wat is dan $f_{\mathrm{E}}(x)$ en $f_{\mathrm{O}}(x)$?