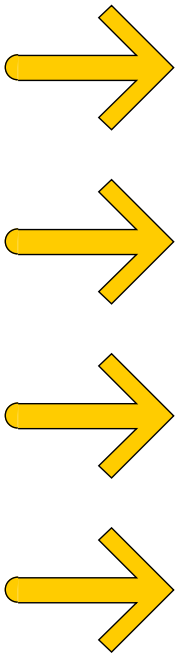
2 Programmeren

Uitdrukkingen ...



waarde van...

toevoegen van variabele...

opzoeken van variabele...

wijzigen van variabele...

...getal
...breuk
...tekst
...tabel
...functie

Evaluatie ...

fout!

$$\underline{\text{read}} : \{\textit{tekst}\} \rightarrow \{\textit{uitdrukking}\} \cup \varnothing$$

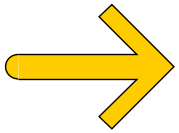
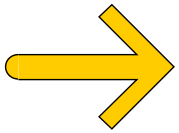
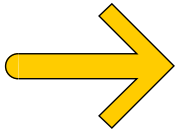
$$\underline{\text{eval}} : \{\textit{uitdrukking}\} \rightarrow \{\textit{uitdrukking}\}$$

$$\underline{\text{print}} : \{\textit{uitdrukking}\} \rightarrow \{\textit{tekst}\}$$

functies van de PICO-machine worden onderstreept

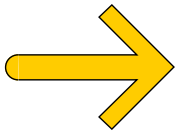
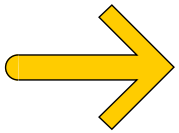
dit noemt men de waarde van een geëvalueerde uitdrukking

Waarden ...

getallen
breuken
tekst

} expliciete notatie

tabellen
functies

} impliciete notatie

Getal waarden ...

- read("{0,1,...,9}$^+$") → *getal*
- eval (*getal*) → *getal*
- print (*getal*) → {0,1,...,9}$^+$

```
getal: read("123")
:123
eval(getal)
:123
print(getal)
:123
```

transcript

V$^+$ betekent
minstens één uit V

Breuk waarden ...

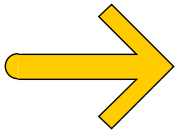
- read("{ 0,1,2,... }$^+$. { 0,1,2,... }$^+$") $\rightarrow$ *breuk*
- eval (*breuk*) $\rightarrow$ *breuk*
- print (*breuk*) $\rightarrow$ { 0,1,2,... }$^+$. { 0,1,2,... }$^+$

```
breuk: read("123.45")
:123.450000
eval(breuk)
:123.450000
print(breuk)
:123.450000
```

transcript

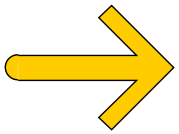
V$^+$ betekent
minstens één uit V

Tekstuele waarden ...

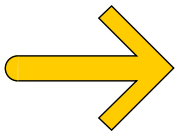


read(" '{ *symbool*}* ' ") → *tekst*

read(' "{ *symbool*}* " ') → *tekst*



eval (*tekst*) → *tekst*



print (*tekst*) → {*symbool*}*

```
tekst: read("'abc'")
:abc
eval(tekst)
:abc
print(tekst)
:abc
```

transcript

V* betekent
minstens nul uit V

Woorden ...

$$\text{naam} \equiv \{a,A,b,B,...\}\{a,A,b,B,...,0,1,2,...\}^*$$

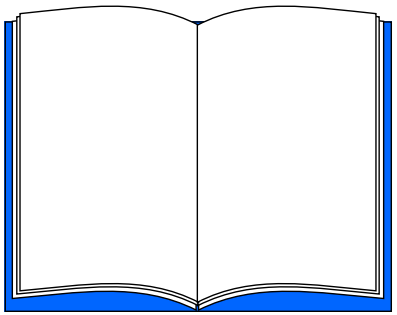
...of...

$$\text{operator} \equiv \{+,-,*,/,\backslash,\hat{},=,<,>,\S,\#,\%,\&,..,|\}^+$$

V^* betekent
minstens nul uit V

V^+ betekent
minstens één uit V

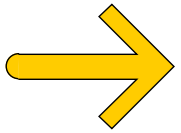
Woordenboeken ...



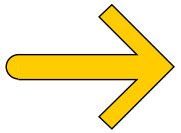
- toevoegen...
- opzoeken...
- vervangen...

} ...van een woord

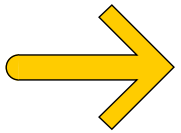
Toevoegen ...



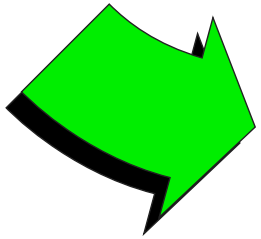
het *woord* wordt achteraan het woordenboek *W* toegevoegd



aan dit *woord* wordt de *waarde* gekoppeld



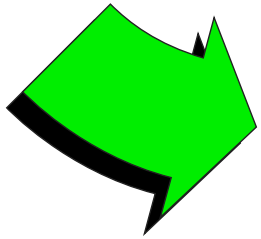
de *uitgebreide W* wordt teruggegeven



add (*woord*, *waarde*, *W*) → *W*

Opzoeken ...

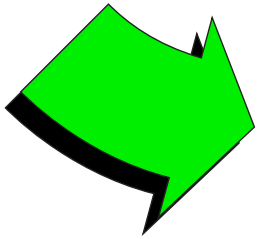
- het *woord* wordt van achter naar voor opgezocht in *W*
- indien niet gevonden is er een fout
- indien wel gevonden wordt de *waarde* gekoppeld aan het *woord* teruggegeven



get(*woord*, *W*) → *waarde*

Vervangen ...

- het *woord* wordt van achter naar voor opgezocht in *W*
- indien niet gevonden is er een fout
- zoniet wordt aan het *woord* de nieuwe *waarde* gekoppeld en wordt void teruggegeven



set(*woord*,*waarde*,*W*) → void

Mogelijke uitwerking ...

```
{add(word,value,dict): [word,value,dict];
 get(word,dict):
   if(is_void(dict),
      display('undefined identifier: ',word),
      if(dict[1]=word,dict[2],get(word,dict[3])));
 set(word,value,dict):
   if(is_void(dict),
      display('undefined identifier: ',word),
      if(dict[1]=word,
         {dict[2]:=value;
          void},
         set(word,value,dict[3])))}
:<function set>
{D: add('abc',123,void);
 D:=add('xyz',789,D);
 D:=add('klm',567,D)}
:[klm, 567, [xyz, 789, [abc, 123, <void>]]]
get('xyz',D)
:789
set('xyz',0,D)
:<void>
get('xyz',D)
:0
```

zie later!

Gebruik van variabelen ...

definitie ...	«*woord* : *uitdrukking*»
verwijzing ...	«*woord*»
wijziging ...	«*woord* := *uitdrukking*»

we noteren Ω voor het globale woordenboek

we noteren read("...") als «...»

Definitie van variabelen ...

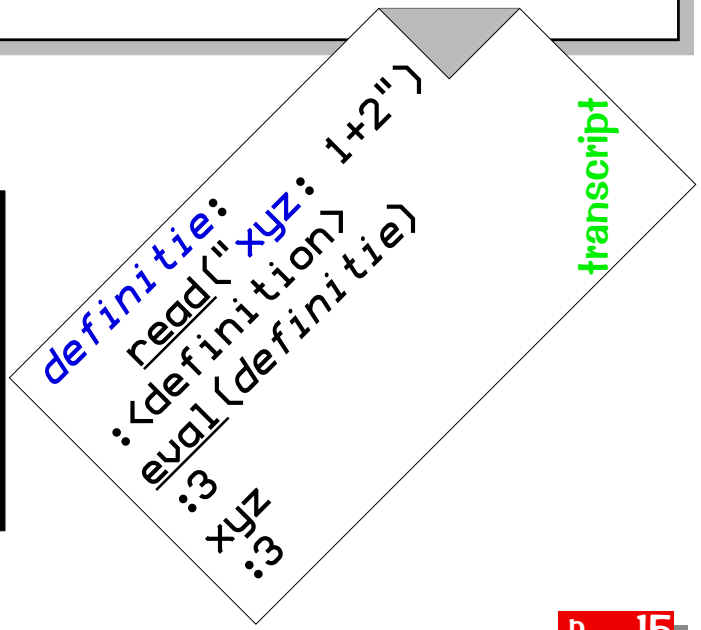
eval(«*woord* : *uitdrukking*»)

≡

{val : eval(«*uitdrukking*»);
Ω:= add(«*woord*»,val,Ω);
val}

de evaluatie van «*woord* : *uitdrukking*» betekent:
1) de evaluatie van «*uitdrukking*»
2) de toevoeging van «*woord*» en de waarde uit 1) aan Ω
3) teruggeven van de waarde uit 1)

dit heet de definitie van een variabele



Verwijzing naar variabelen ...

eval(«*woord*»)

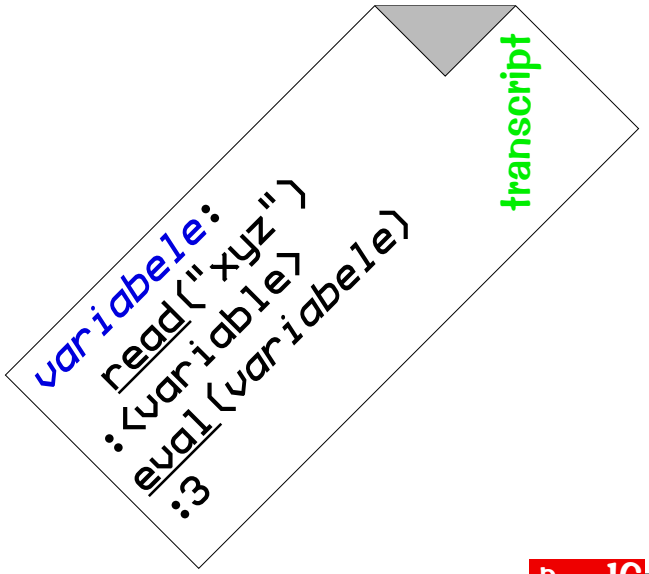
≡

get(«*woord*»,Ω)

de evaluatie van «*woord*» betekent:

1) de opvraging van «*woord*» aan Ω
2) teruggeven van de waarde uit 1)

dit heet de verwijzing naar een variabele



Wijziging van variabelen ...

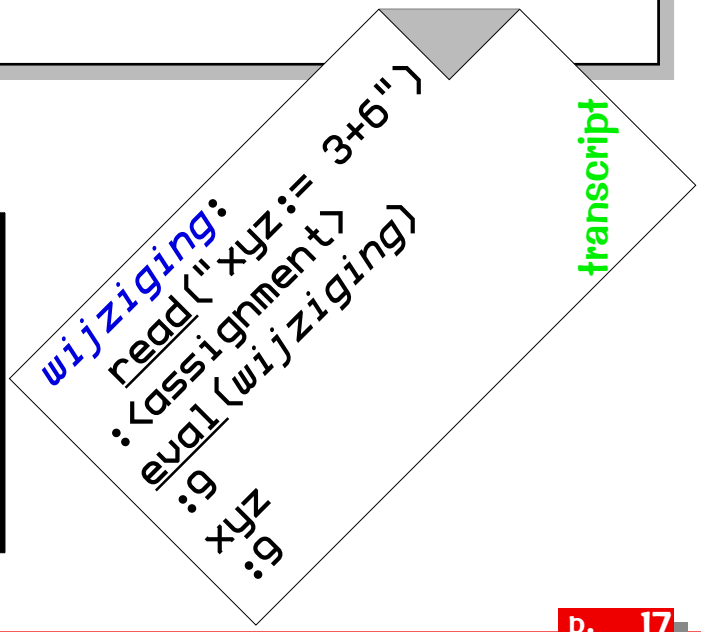
eval(«*woord* := *uitdrukking*»)

≡

{ val : eval(«*uitdrukking*»);
  set(«*woord*»,val,Ω);
  val}

de evaluatie van «*woord* := *uitdrukking*» betekent:
1) de evaluatie van «*uitdrukking*»
2) de vervanging van «*woord*» met de waarde uit 1) in Ω
3) teruggeven van de waarde uit 1)

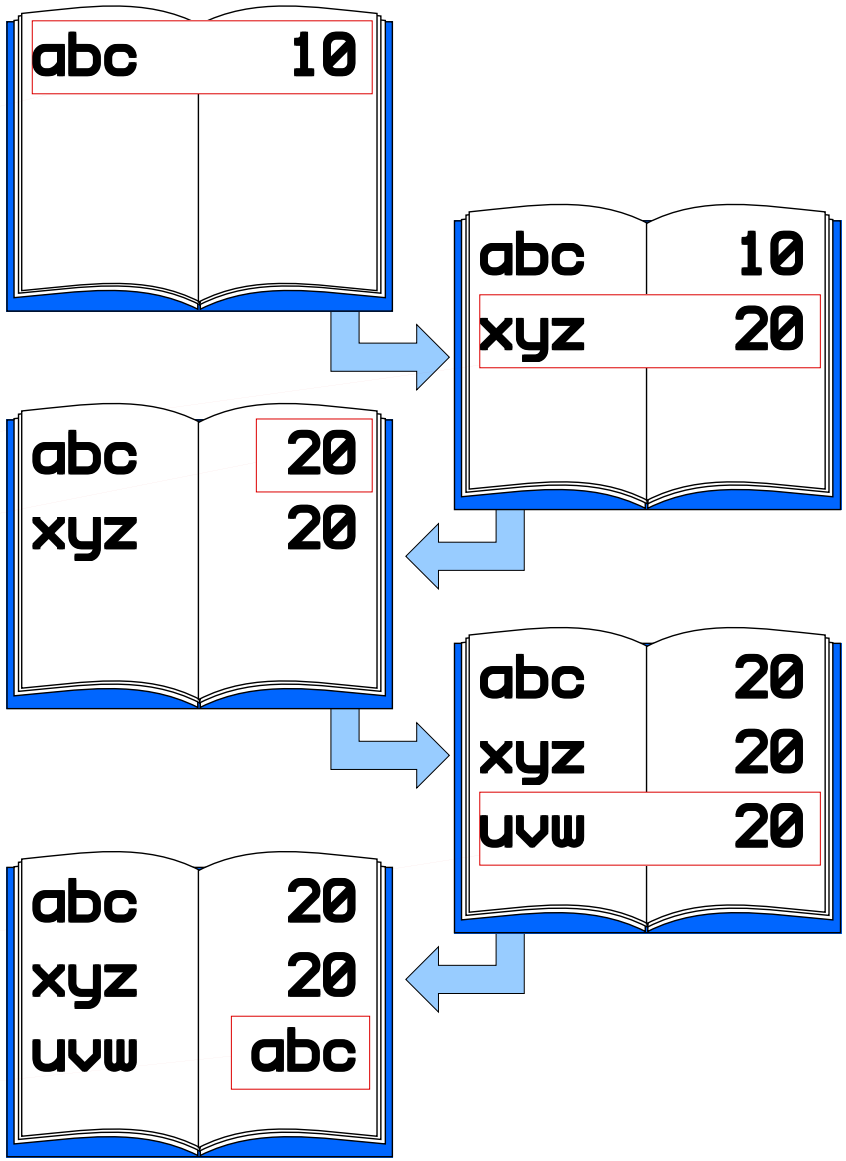
dit heet de wijziging van een variabele



Voorbeeld ...

```
abc: 10
:10
xyz:= 20
:undefined identifier: xyz
xyz: 20
:20
abc:= xyz
:20
abc
:20
uvw
:undefined identifier: uvw
uvw: abc
:20
uvw:= 'abc'
:abc
```

transcript



Gebruik van functies ...

argument = uitdrukking

definitie ... «*woord*(*parameter*,...) : *uitdrukking*»

toepassing ... «*woord*(*argument*,...)»

wijziging ... «*woord*(*parameter*,...) := *uitdrukking*»

parameter = *woord*#

X,... betekent 0 of méér elementen X gescheiden door komma's

#voorlopig

Definitie van functies ...

om recursie te ondersteunen

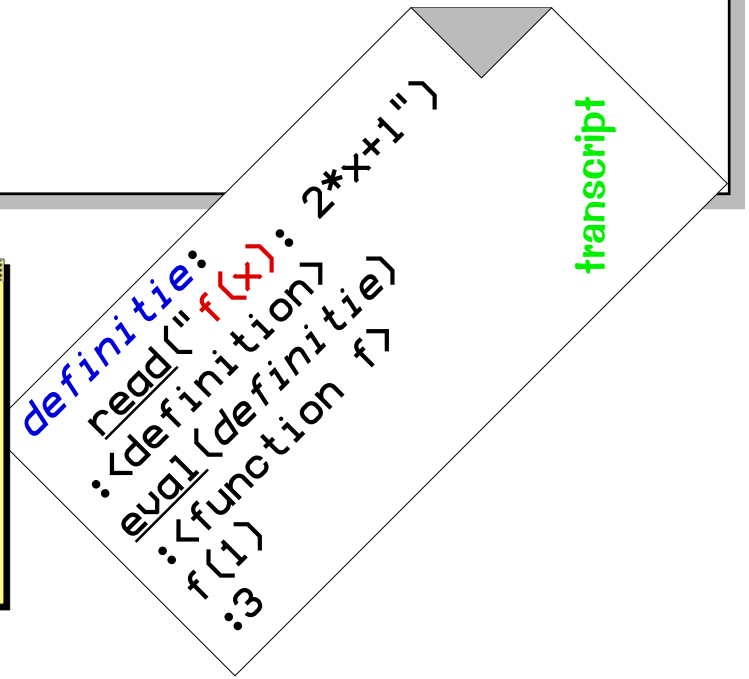
eval(«*woord*(*parameter*,...): *uitdrukking*»)

≡

{Ω := add(«*woord*»,void,Ω);
fun: [[«*parameter*»,...],«*uitdrukking*»,Ω];
set(«*woord*»,fun,Ω);
fun}

de evaluatie van «*woord*(*parameter*, ...) : *uitdrukking*» betekent:
1) de aanmaak van een functiewaarde aan de hand van: [«*parameter*», ...], «*uitdrukking*» en Ω
2) de binding van de waarde uit 1) aan «*woord*» binnen Ω
3) opleveren van de waarde uit 1)

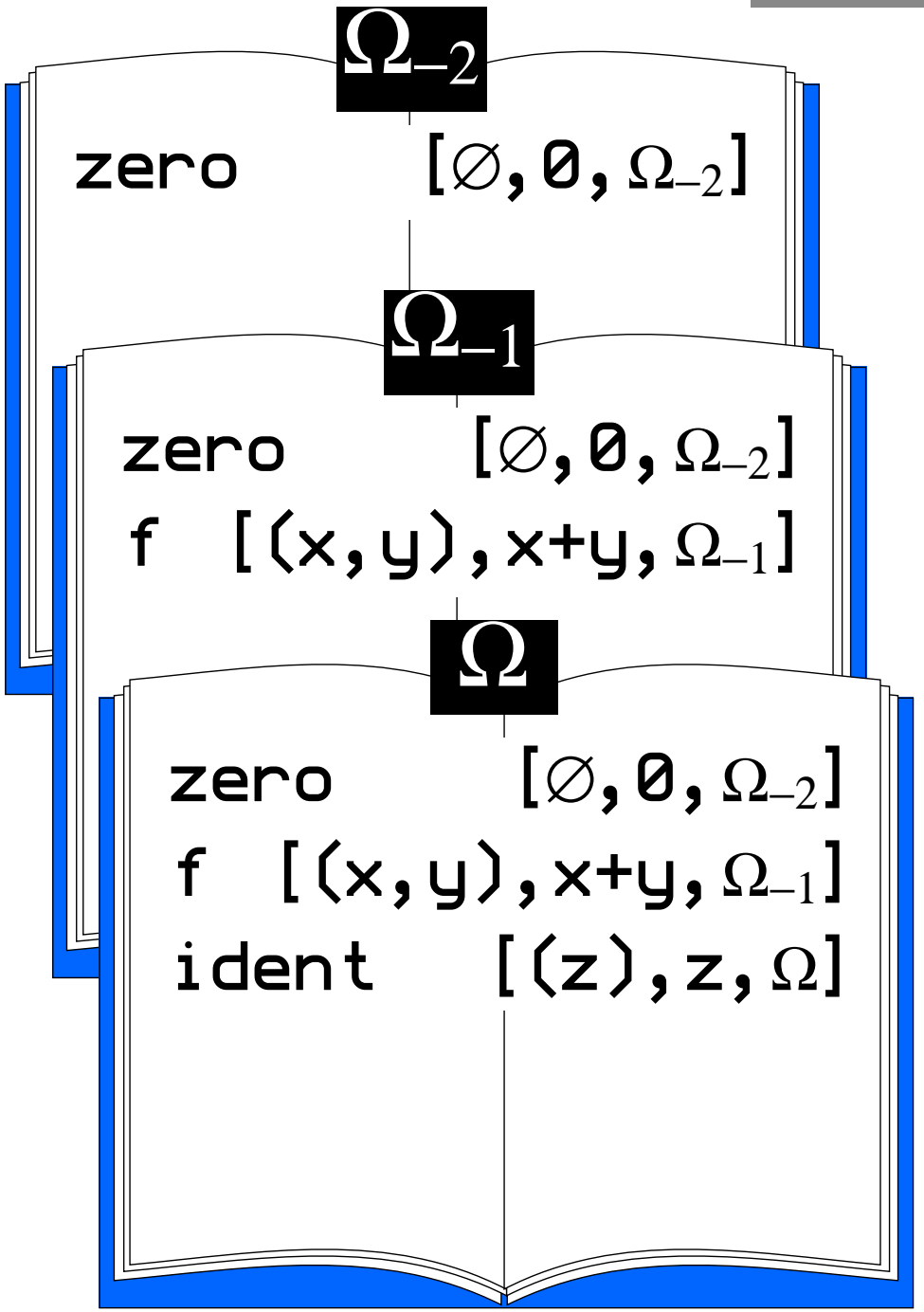
dit heet de definitie van een functie



Voorbeeld ...

```
zero(): 0
:<function zero>
f(x,y): x+y
:<function sum>
ident(z): z
:<function ident>
```

transcript

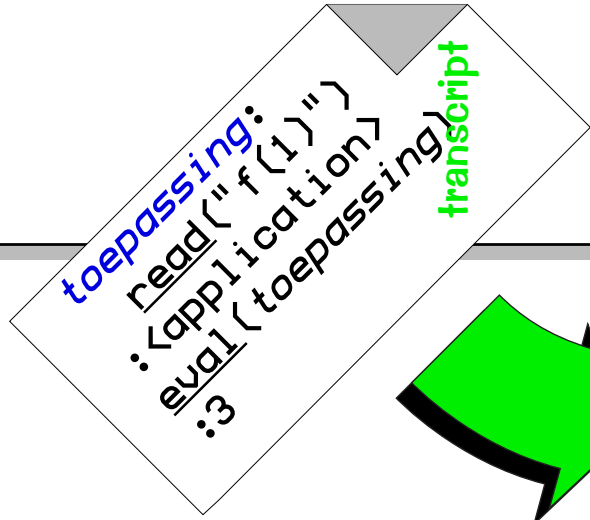


Toepassing van functies ...

eval(«*woord*(*argument*,...)»)

≡

```
{fun: get(«woord»,Ω):
  dict: Ω;
  Ω:= bind([«argument»,...],fun[1],fun[3]);
  val: eval(fun[2]):
  Ω:= dict;
  val}
```



dit heet de toepassing van een functie

Toepassing van functies (#1)

fun: get(«*woord*»,Ω)

- «*woord*» wordt opgezocht in Ω
- het resultaat wordt bewaard in fun
- indien fun geen functiewaarde bevat wordt een fout gemeld

Toepassing van functies (#2)

dict: Ω

het woordenboek Ω wordt bewaard in dict

Toepassing van functies (#3)

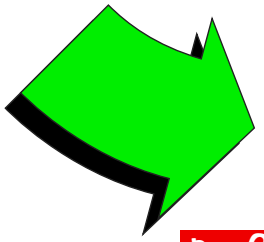
Ω:= bind([«*argument*»,...],fun[1],fun[3])

bind([«*argument*»,...],[«*parameter*»,...],dict)

≡

bind(«*argument*»,«*parameter*»,dict) ...

bind maakt een binding tussen de argumenten en de parameters binnen het woordenboek van de functie



Binding van parameters ...

bind(«*argument*»,«*parameter*»,dict)

≡

dict:= add(«*parameter*»,eval(«argument»),dict)

de binding van «*argument*» aan «*parameter*» betekent:

1) indien «*parameter*» geen woord is wordt een fout veroorzaakt#
2) de evaluatie van «*argument*» binnen dict
3) de toevoeging van «*parameter*» en de waarde uit 2) aan dict

= voorlopige beperking

Toepassing van functies (#4)

val: eval(fun[2])
≡
val: eval(«uitdrukking»)

de waarde van de functieuitdrukking
wordt bepaald en bewaard in val

Toepassing van functies (#5) ...

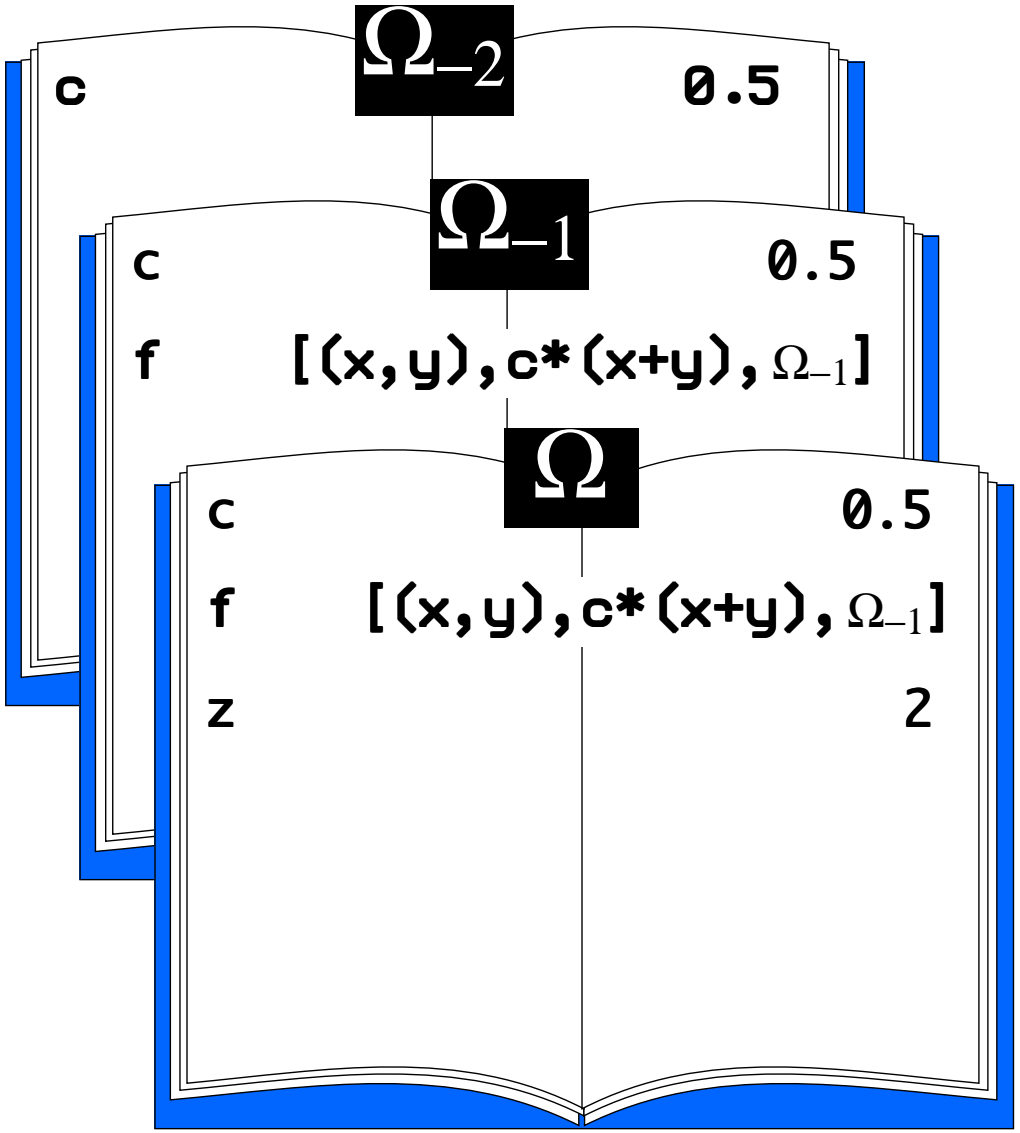
```
Ω:= dict;
val
```

het oorspronkelijke woordenboek wordt hersteld vanuit dict en de waarde in val wordt teruggegeven als waarde van de funcietoepassing

Functietoepassing: voorbeeld 1...

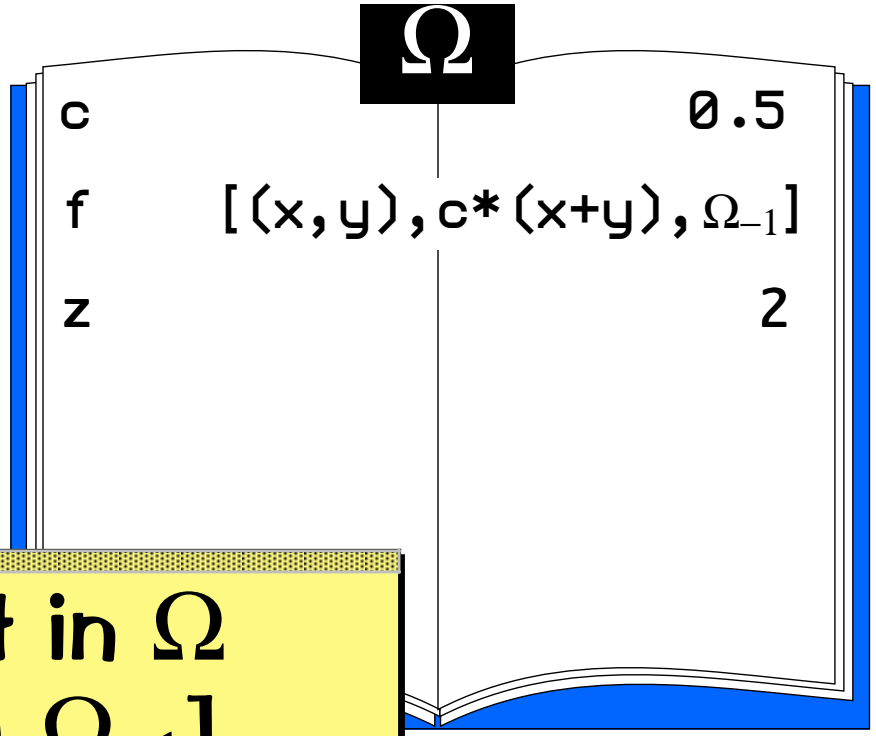
```
c:0.5
:0.50000
f(x,y): c*(x+y)
:<function f>
z:2
:2
f(1+2,z)
```

transcript



Functietoepassing: voorbeeld 1...

f(1+2,z)

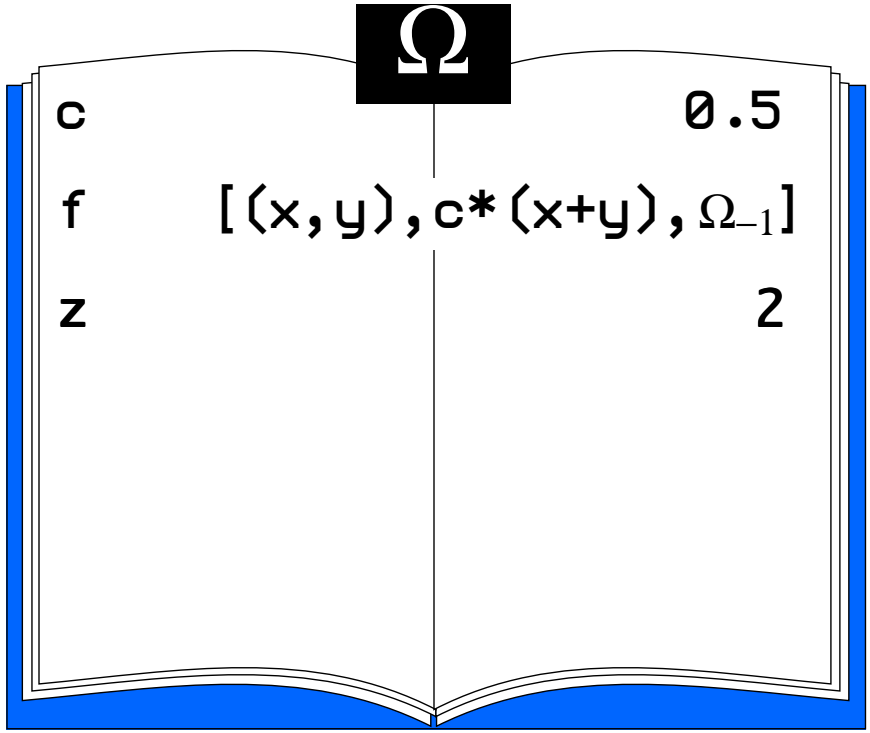


stap 1: f wordt opgezocht in Ω
⇒ fun := [(x,y),c*(x+y),Ω−1]

stap 2

Functietoepassing: voorbeeld 1...

```
f(1+2,z)
```

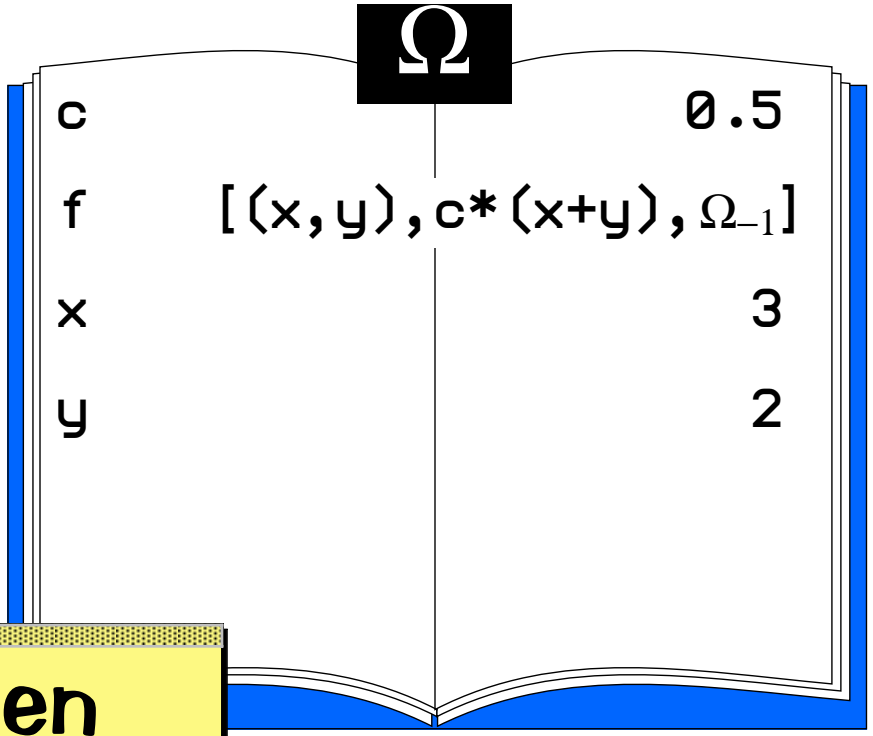


stap 2: Ω wordt bewaard in dict

stap 3

Functietoepassing: voorbeeld 1...

f(1+2,z)

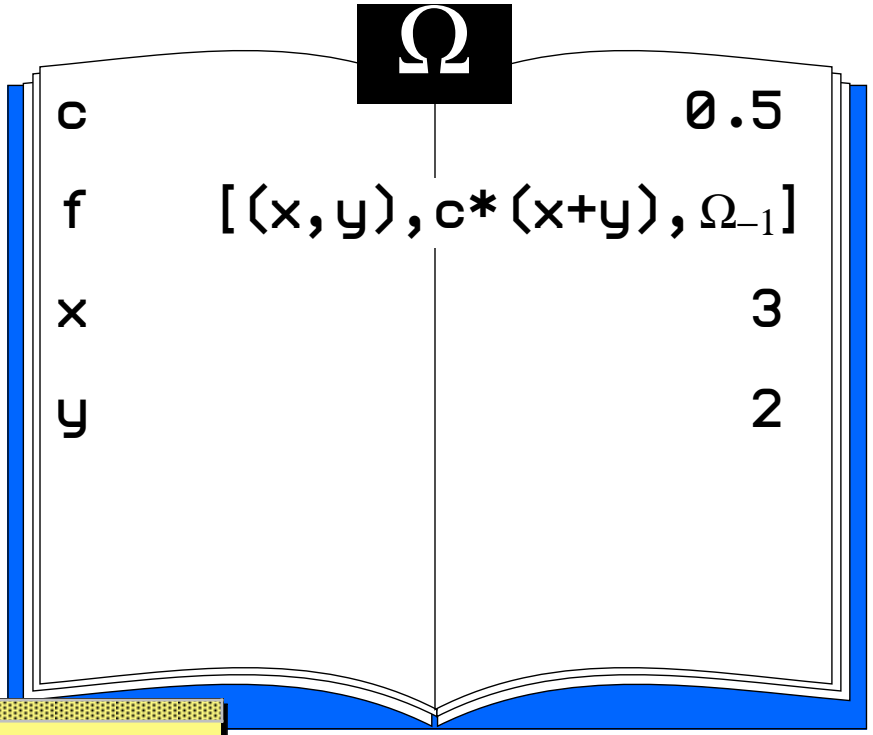


stap 3: Ω wordt vervangen door de binding van (1+2,z) aan (x,y) binnen Ω_{-1}

stap 4

Functietoepassing: voorbeeld 1...

f(1+2,z)

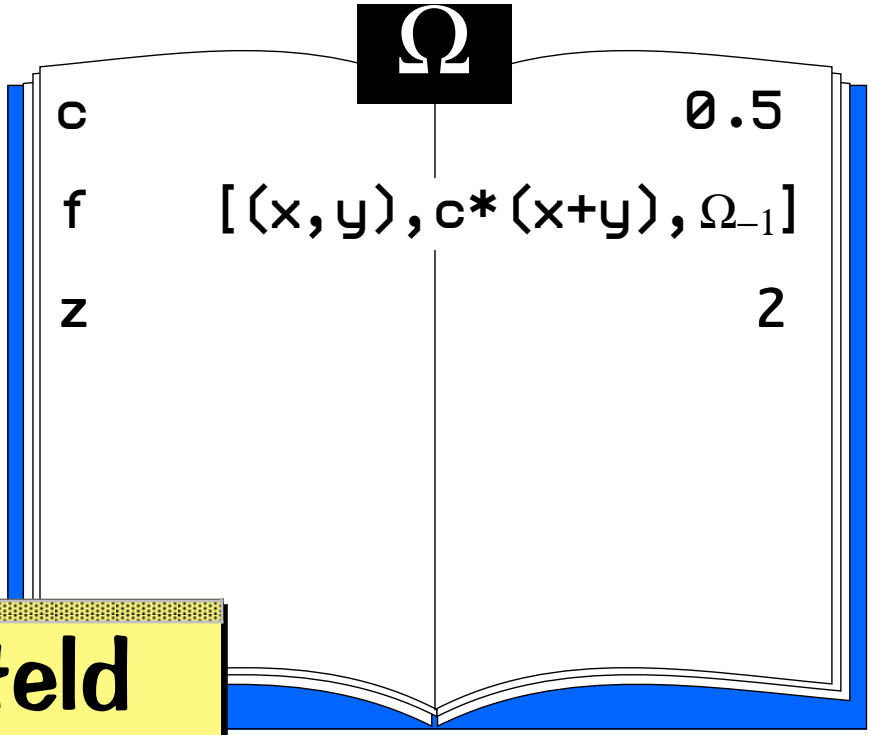


stap 4: de waarde van c*(x+y) wordt bewaard in val

stap 5

Functietoepassing: voorbeeld 1...

f(1+2,z)

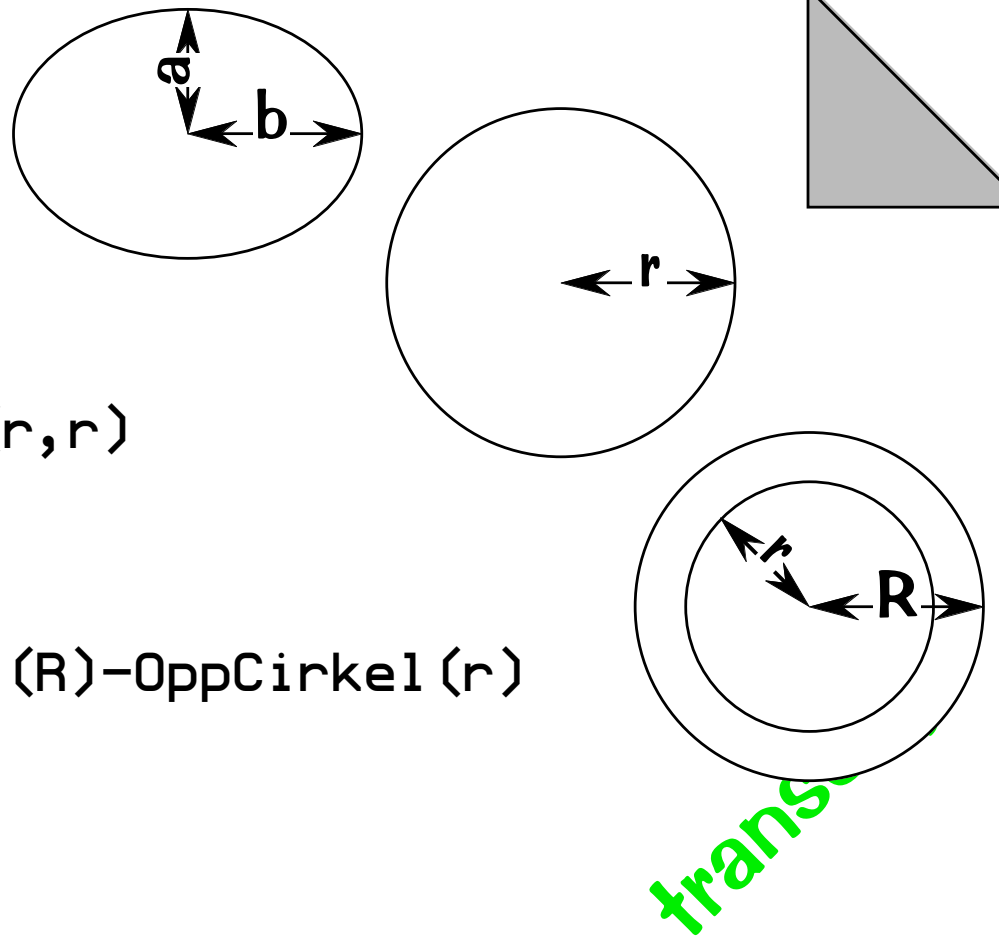


stap 5: Ω wordt hersteld vanaf dict en de waarde van val wordt teruggegeven

2.5

Functietoepassing: voorbeeld 2...

```
Pi: 3.141593
:3.141593
OppEllips(a,b): Pi*a*b
:<function OppEllips>
OppEllips(2,3)
:18.849556
OppCirkel(r): OppEllips(r,r)
:<function OppCirkel>
OppCirkel(1)
:3.141593
OppDonut(R,r): OppCirkel(R)-OppCirkel(r)
:<function OppDonut>
OppDonut(2,1)
:9.424778
```



Wijziging van functies ...

eval(«*woord*(*parameter*,...):= *uitdrukking*»)

≡

{fun: [[«*parameter*»,...],«*uitdrukking*»,Ω];
set(«*woord*»,fun,Ω);
fun}

de evaluatie van «*woord*(*parameter*, ...) := *uitdrukking*» betekent:

1) de aanmaak van een functiewaarde aan de hand van: [«*parameter*», ...], «*uitdrukking*» en Ω
2) de toekenning van de waarde uit 1) aan «*woord*» binnen Ω
3) opleveren van de waarde uit 1)

dit heet de wijziging van een functie

```
wijziging:
  read("f(x):= 2*x-1")
 :<assignment>
  eval(wijziging)
 :<function f>
  f(1)
 :1
```

transcript

Operatornotatie ...

$$\bullet(W): U \equiv \bullet W: U$$

$$\bullet(U) \equiv \bullet U$$

$$\bullet(W_1, W_2): U \equiv W_1 \bullet W_2: U$$

$$\bullet(U_1, U_2) \equiv U_1 \bullet U_2$$

unaire operaties

binaire operaties

Voorrang bij unaire operaties ...

```
+123
:123
-12
:-12
-+2
:undefined identifier: -+
-(+2)
:-2
-1+2
:1
-(1+2)
:-3
```

transcript

niet zo

maar zo!

verschillend!

Voorrang bij binaire operaties ...

$$u_1 \varphi u_2 \psi u_3 \equiv \begin{cases} (u_1 \varphi u_2) \psi u_3 & \text{als } \psi \leq \varphi \\ u_1 \varphi (u_2 \psi u_3) & \text{als } \psi > \varphi \end{cases}$$

...>...
betekent:
"heeft voorrang op"

Voorrang bij binaire operaties (vervolg) ...

X = { ! ? ^ }	exponentiële	operaties
M = { & * / \ }	multiplicatieve	
A = { $ % + - | ~ }	additieve	
R = { # < = > }	relationele	

X > M > A > R

φ > ψ indien
- initiaal(φ) ∈ V
- initiaal(ψ) ∈ W
- V > W

Voorrang bij binaire operaties (vervolg) ...

```
a++b: a+b+1
:<function ++>
a**b: a*b*2
:<function **>
a^#b: a^(b/2)
:<function ^#>
p<=>q: abs(p-q)<1
:<function <=>>
1++2<=>1**2
:<native function true>
(1**1)^#(1++1)
:2.828427
1**1^#(1++1)
:2.000000
1**1^#1++1
:4.000000
```

transcript

1**2 = 1*2*2 = 4
1++2 = 1+2+1 = 4
4<=>4 = abs(4-4) < 1 = true

1**1 = 1*1*2 = 2
1++1 = 1+1+1 = 3
2^#3 = 2^1.5 = 2.828427

1++1 = 1+1+1 = 3
1^#3 = 1^1.5 = 1.000000
1**1.000000 = 2.000000

1^#1 = 1^0.5 = 1.000000
1**1.000000 = 2.000000
2.000000++1 = 4.000000

Voorgedefinieerde rekenkundige functies ...

+p:	...numerieke identiteit...
-p:	...negatie van p...
p+q:	...som van p en q...
p-q:	...verschil van p en q...
p*q:	...product van p en q...

+ mag ook gebruikt worden voor de samenstelling van tekst: zie later

het resultaat is een breuk indien één of beide argumenten een breuk is, zoniet is het geheel

Voorgedefinieerde rekenkundige functies (vervolg)...

p//q: ...quotiënt van p en q...
p\\q: ...rest van p en q...

beide argumenten moeten geheel zijn; het resultaat is dat ook

p/q: ...deling van p en q...
p^q: ...p tot de macht van q...

het resultaat is steeds een breuk

Voorgedefinieerde rekenkundige functies (voorbeeld)...

```
!n: if(n>1,!(n-1)*n,1)
:<function !>
!10
:3628800
C(p,q): !p//(!q*(!(p-q)))
:<function C>
C(10,5)
:252
Bin(k):
   {hulp(n):
      {display(C(k,n),', ');
       if(n<k,hulp(n+1),'ok')};
    hulp(0)}
:<function Bin>
Bin(10)
:1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1, ok
```

$$C_q^p = \frac{p!}{q!(p-q)!}$$

transcript

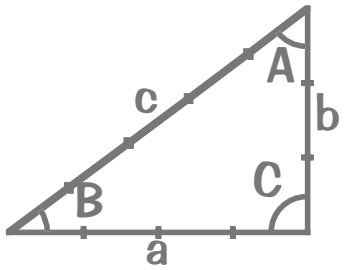
Voorgedefinieerde transcedente functies ...

sin(p):	...sinus van p...
cos(p):	...cosinus van p...
tan(p):	...tangens van p...
arcsin(p):	...boogsinus van p...
arccos(p):	...boogcosinus van p...
arctan(p):	...boogtangens van p...
sqrt(p):	...wortel van p...
exp(p):	...exponentiaal van p...
log(p):	...logaritme van p...

het resultaat is steeds een breuk

Voorgedefinieerde transcendente functies (voorbeeld) ...

```
CosinusRegel(a,b,C): sqrt(a^2+b^2+2*a*b*cos(C))
:<function CosinusRegel>
SinusRegel(a,A,C): a*sin(C)/sin(A)
:<function SinusRegel>
Pi: 3.14159265358979
:3.141592
CosinusRegel(4,3,Pi/2)
:5.000000
SinusRegel(4,arctan(4/3),Pi/2)
:5.000000
```



transcript

Voorgedefinieerde logische functies ...

```
true(x,y):  ...waarde van x...
false(x,y): ...waarde van y...
and(p,q):   p(q,false)
or(p,q):    p(true,q)
not(p):     p(false,true)
```

formele definitie volgt later

Voorgedefinieerde logische functies (voorbeeld) ...

```
true
:<native function true>
true(display('ja'),display('neen'))
:ja
false(display('ja'),display('neen'))
:neen
WaarheidsTabel(op):
   {display('T <op> T => ',if(op(true ,true ),'T','F'),eoln);
    display('T <op> F => ',if(op(true ,false),'T','F'),eoln);
    display('F <op> T => ',if(op(false,true ),'T','F'),eoln);
    display('F <op> F => ',if(op(false,false),'T','F'),eoln)}
:<function WaarheidsTabel>
```

transcript

Voorgedefinieerde logische functies (voorbeeld) ...

```
Verschil(p,q): and(p,not(q))
:<function Verschil>
WaarheidsTabel(Verschil)
:T <op> T => F
:T <op> F => T
:F <op> T => F
:F <op> F => F
:
SymVersch(p,q): Verschil(or(p,q),and(p,q))
:<function SymVersch>
WaarheidsTabel(SymVersch)
:T <op> T => F
:T <op> F => T
:F <op> T => T
:F <op> F => F
:
```

transcript

Voorgedefinieerde relationele functies ...

x=y: ...antwoord met true of false...
x>y: ...antwoord met true of false...
x<y: ...antwoord met tr...

de argumenten moeten allebei ofwel getallen, ofwel tekst zijn

equivalent(x,y): ...antwoord met true of false...

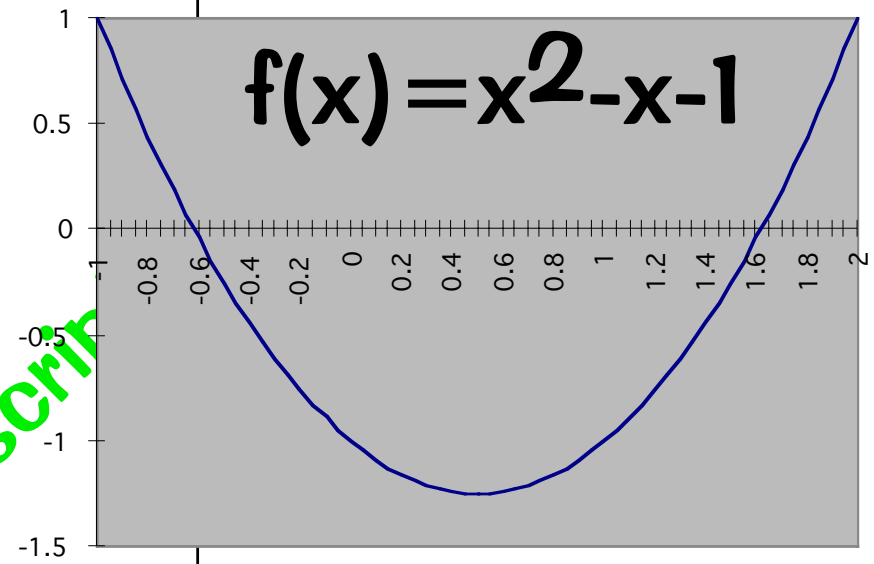
de argumenten mogen willekeurig zijn

void: ...nihil...

universele lege waarde

Voorgedefinieerde relationele ... (voorbeeld)...

```
RegulaFalsi(f,x0,x1):
   {x: (x0+x1)/2;
    y: f(x);
    if(abs(y)<1e-6,
       x,
       if(y*f(x0)<0,
          RegulaFalsi(f,x0,x),
          RegulaFalsi(f,x,x1)))}
:<function RegulaFalsi>
test(x): x^2-x-1
:<function test>
RegulaFalsi(test,1,2)
:1.618034
(1+sqrt(5))/2
:1.618034
RegulaFalsi(test,-1,0)
:-0.618034
(1-sqrt(5))/2
:-0.618034
```



transcrip

Voorgedefinieerde relationele functies (voorbeeld)...

```
RegulaFalsi(f,x0,x1):
    {x: (x0+x1)/2;
     y: f(x);
     if(abs(y)<1e-6,
        x,
        if(y*f(x0)<0,
           RegulaFalsi(f,x0,x),
           RegulaFalsi(f,x,x1)))}
```

middelpunt

functiewaarde in het middelpunt

indien "wel 0"

indien "niet 0"

indien het teken tussen x0 en x verandert

indien het teken tussen x en x1 verandert

Voorgedefinieerde relationele (voorbeeld)...

```
equivalent(1,1.0)
:<native function false>
equivalent('abc','abc')
:<native function false>
x: 'abc'
:abc
y: x
:abc
equivalent(x,y)
:<native function true>
f(x): x
:<function f>
g(): 0
:<function g>
equivalent(f,g)
:<native function false>
z: void
:<void>
equivalent(z,void)
:<native function true>
```

transcript

twee waarden zijn equivalent indien ze afstammen van hetzelfde resultaat van eval

Voorgedefinieerde invoer/uitvoer functies ...

```
display(x,y,...):
    {print(x); print(y); ...}
```

```
accept(): ...lees een regel in...
```

```
eoln: ...einde regel...
```

alle ingegeven tekens tot de "enter"-toets wordt gebruikt

Voorgedefinieerde invoer/uitvoer (voorbeeld) ...

```
vulling(n,c):
  if(n>0,
     {display(c);
      vulling(n-1,c)},
     void)
:<function vulling>
etiketten(breedte,kolommen,rijen):
  {lees(tekst):
     {display('geef ',tekst,': ');
      x: accept();
      display(eoln);
      x};
   naam: lees('naam');
   voornaam: lees('voornaam');
   adres: lees('adres');
   postnr: lees('postnummer');
   gemeente: lees('gemeente');
   n1: breedte-size(voornaam)-size(naam)-1;
   n2: breedte-size(adres);
   n3: breedte-size(postnr)-size(gemeente)-1;
```

functie die *n* keer de inhoud van *c* uitschrijft

functie die met *tekst* aanspoort om iets in te lezen

hier worden de gegevens voor één etiket ingelezen

hier wordt de "lege plaats" voor elke regel van een etiket berekend

...er/uitvoer ...eeld) ...

```
f1(): display(voornaam,' ',naam);
f2(): display(adres);
f3(): display(postnr,' ',gemeente);
kolom(f,n,k):
   {f();
    if(k>1,
       {vulling(n,' ');
        kolom(f,n,k-1)},
       display(eoln))};
rij(r):
   {display(eoln,eoln);
    kolom(f1,n1,kolommen);
    kolom(f2,n2,kolommen);
    kolom(f3,n3,kolommen);
    if(r>1,rij(r-1),display(eoln,eoln))};
vulling(breedte*kolommen,'-');
display(eoln);
rij(rijen);
vulling(breedte*kolommen,'-');
display(eoln)}
:<function etiketten>
```

hulpfuncties om telkens één regel van een etiket uit te schrijven

functie om per rij een aantal kolommen te schrijven

functie om een aantal rijen etiketten te schrijven

transcript

Voorgedefinieerde invoer/uitvoer

...

```
etiketten(25,4,4)

:geef naam: D'Hondt
:geef voornaam: Theo
:geef adres: Raambeekweg 11
:geef postnummer: 3140
:geef gemeente: Keerbergen
:-------------------------------------------------------------------------------------
:
:
:Theo D'Hondt             Theo D'Hondt             Theo D'Hondt             Theo D'Hondt
:Raambeekweg 11           Raambeekweg 11           Raambeekweg 11           Raambeekweg 11
:3140 Keerbergen          3140 Keerbergen          3140 Keerbergen          3140 Keerbergen
:
:
:Theo D'Hondt             Theo D'Hondt             Theo D'Hondt             Theo D'Hondt
:Raambeekweg 11           Raambeekweg 11           Raambeekweg 11           Raambeekweg 11
:3140 Keerbergen          3140 Keerbergen          3140 Keerbergen          3140 Keerbergen
:
:
:Theo D'Hondt             Theo D'Hondt             Theo D'Hondt             Theo D'Hondt
:Raambeekweg 11           Raambeekweg 11           Raambeekweg 11           Raambeekweg 11
:3140 Keerbergen          3140 Keerbergen          3140 Keerbergen          3140 Keerbergen
:
:
:Theo D'Hondt             Theo D'Hondt             Theo D'Hondt             Theo D'Hondt
:Raambeekweg 11           Raambeekweg 11           Raambeekweg 11           Raambeekweg 11
:3140 Keerbergen          3140 Keerbergen          3140 Keerbergen          3140 Keerbergen
:
:
:-------------------------------------------------------------------------------------
:
:
```

transcript

Voorgedefinieerde conversie functies ...

trunc(x):	...gehele waarde van *x*...
abs(x):	...absolute waarde van *x*...
char(x):	...tekst bestaande uit één letterteken met volgnummer *x*...
ord(x):	...volgnummer van *x* bestaande uit één letterteken...
number(x):	...tekst *x* als getal of breuk...
text(x):	...waarde van *x* als tekst...

Voorgedefinieerde conversie functies (voorbeeld 1) ...

```
{display('geef een getal>');
 txt: accept();
 nbr: number(txt);
 if(is_number(nbr),
    {bin(nbr): if(nbr<2,
                  display(nbr),
                  {bin(nbr//2);
                   display(nbr\\2)});
     display(eoln,nbr,'DEC = ');
     bin(nbr);
     display('BIN')},
    display(' dit is geen getal!',eoln))}
:geef een getal>1234
:1234DEC = 10011010010BIN
```

transcript

Voorgedefinieerde conversie functies (voorbeeld 1) ...

```
{display('geef een getal>');
 txt: accept();
 nbr: number(txt);
 if(is_number(nbr),
    {bin(nbr): if(nbr<2,
                  display(nbr),
                  {bin(nbr//2);
                   display(nbr\\2)});
     display(eoln,nbr,'DEC = ');
     bin(nbr);
     display('BIN')},
    display(' dit is geen getal!',eoln))}
```

lees een regel in

zet de ingelezen tekst om naar een getal

controleer dat het een geldig getal is

functie die de binaire omzetting doet

eindvoorwaarde voor de recursie

behandel eerst de begincijfers

schrijf dan het laatste cijfer uit

Voorgedefinieerde conversie

```
ascii():
  {hoofding(n): if(n<16,
                   {display(if(n<10,'  ',' '),n);
                    hoofding(n+1)},
                   display(eoln,eoln));
   rij(n):
     {kolom(m,n): if(m<16,
                     {display(' ',char(n),' ');
                      kolom(m+1,n+1)},
                     display(eoln));
      if(n<256,
         {display(if(n<10,'  ',if(n<100, ' ', ' ')),n,'  ');
          kolom(0,n);
          rij(n+16)},
         display(eoln))};
   display('    ');
   hoofding(0);
   rij(32)}
:<function ascii>
```

transcript

Voorgedefinieerde conversie functies (voorbeeld)

```
ascii()
:       0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
:
:  32      !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
:  48   0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
:  64   @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
:  80   P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _
:  96   `  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
: 112   p  q  r  s  t  u  v  w  x  y  z  {  |  }  ~
: 128   Ä  Å  Ç  É  Ñ  Ö  Ü  á  à  â  ä  ã  å  ç  é  è
: 144   ê  ë  í  ì  î  ï  ñ  ó  ò  ô  ö  õ  ú  ù  û  ü
: 160   †  °  ¢  £  §  •  ¶  ß  ®  ©  ™  ´  ¨  ≠  Æ  Ø
: 176   ∞  ±  ≤  ≥  ¥  µ  ∂  Σ  Π  π  ∫  ª  º  Ω  æ  ø
: 192   ¿  ¡  ¬  √  ƒ  ≈  Δ  «  »  …     À  Ã  Õ  Œ  œ
: 208   –  —  “  ”  ‘  ’  ÷  ◊  Ÿ  Ÿ  ⁄  ¤  ‹  ›  fi  fl
: 224   ‡  ·  ‚  „  ‰  Â  Ê  Á  Ë  È  Í  Î  Ï  Ì  Ó  Ô
: 240      Ò  Ú  Û  Ù  ı  ˆ  ˜  ¯  ˘  ˙  ˚  ¸  ˝  ˛  ˇ
```

transcript

Voorgedefinieerde conversie

```
ascii():
  {hoofding(n): if(n<16,
                   {display(if(n<10,'  ',' '),n);
                    hoofding(n+1)},
                   display(eoln,eoln));
   rij(n):
     {kolom(m,n): if(m<16,
                     {display(' ',char(n),' ');
                      kolom(m+1,n+1)},
                     display(eoln));
      if(n<256,
         {display(if(n<10,'  ',if(n<100, ' ', ' ')),n,'  ');
          kolom(0,n);
          rij(n+16)},
         display(eoln))};
   display('    ');
   hoofding(0);
   rij(32)}
```

Voorgedefinieerde controle functies ...

{uitdrukking;...}

≡

begin(*uitdrukking*,...):
...doe eval(*uitdrukking*),... en
geef de laatste waarde terug

formele definitie:
zie later

Voorgedefinieerde controle functies (voorbeeld) ...

```
x: 123
:123
y: 456
:456
z: 789
:789
begin(s:x,x:=y,y:=z,z:=s,void)
:<void>
display(x,y,z)
:456789123
{s:x; x:=z; z:=y; y:=s; void}
:<void>
display(x,y,z)
:123456789
```

transcript

Voorgedefinieerde controle functies (vervolg) ...

if(*voorwaarde*,*uitdrukking*$_1$,*uitdrukking*$_2$):
...doe eval(*uitdrukking*$_1$) indien
eval(*voorwaarde*)=true, zoniet
doe eval(*uitdrukking*$_2$) ...

formele definitie:
zie later

Voorgedefinieerde controle functies (voorbeeld) ...

```
ggd(x,y):
  if(x>y,
     ggd(x-y,y),
     if(x<y,
        ggd(x,y-x),
        x))
:<function ggd>
ggd(121,33)
:11
```

transcript

Voorgedefinieerde controle functies (vervolg) ...

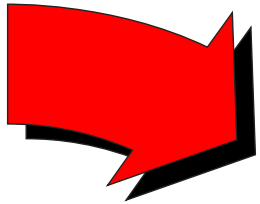
while(*voorwaarde*,*uitdrukking*):
...doe eval(*uitdrukking*)
zolang eval(*voorwaarde*)=true...

formele definitie:
zie later

Voorgedefinieerde controle functies (voorbeeld) ...

```
ggd(x,y):
  if(x>y,
     ggd(x-y,y),
     if(x<y,
        ggd(x,y-x),
        x))
:<function ggd>
ggd(121,33)
:11
```

transcript



```
ggd(x,y):
  while(not(x=y),
        if(x>y,
           x:= x-y,
           y:= y-x))
:<function ggd>
ggd(121,33)
:11
```

transcript

Voorgedefinieerde controle functies (voorbeeld) ...

```
bin(nbr): if(nbr<2,
             display(nbr),
             {bin(nbr//2);
              display(nbr\\2)})
:<function bin>
bin(1234)
:10011010010
```

gaat *niet* met *while*

```
binX(nbr): if(nbr<2,
              display(nbr),
              {display(nbr\\2);
               binX(nbr//2)})
:<function binX>
binX(1234)
:01001011001
```

gaat *wel* met *while* ...

```
binX(nbr): while(not(nbr<2),
                 {display(nbr\\2);
                  nbr:= nbr//2})
:<function binX>
binX(1234)
:01001011001
```

transcript

...maar geeft geen correct resultaat!

Voorgedefinieerde controle functies (vervolg) ...

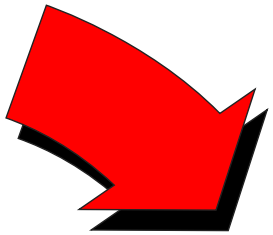
until(*voorwaarde*,*uitdrukking*):
...doe eval(*uitdrukking*)
tot eval(*voorwaarde*)=true...

formele definitie:
zie later

Voorgedefinieerde controle functies (vervolg&voorbeeld) ...

```
RegulaFalsi(f,x0,x1):
   {x: (x0+x1)/2;
    y: f(x);
    if(abs(y)<1e-6,
       x,
       if(y*f(x0)<0,
          RegulaFalsi(f,x0,x),
          RegulaFalsi(f,x,x1)))}
:<function RegulaFalsi>
```

transcript



```
RegulaFalsi(f,x0,x1):
  until(abs(y)<1e-6,
        if((y: f(x: (x0+x1)/2))*f(x0)<0,
           x1:= x,
           x0:= x))
:<function RegulaFalsi>
```

transcript

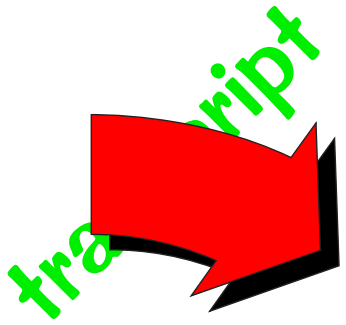
Voorgedefinieerde controle functies (vervolg) ...

```
for(init,incr,voorwaarde,uitdrukking):
      ...doe { waarde: eval(uitdrukking);
               eval(incr);
               waarde}
             zolang eval(voorwaarde)=true...
```

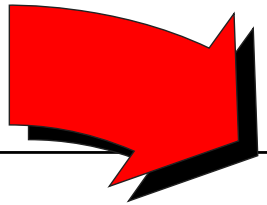
formele definitie:
zie later

Voorgedefinieerde controle

```
ascii():
  {hoofding(n): if(n<16,
                   {display(if(n<10,'  ',' '),n);
                    hoofding(n+1)},
                   display(eoln,eoln));
   rij(n):
     {kolom(m,n): if(m<16,
                     {display(' ',char(n),' ');
                      kolom(m+1,n+1)},
                     display(eoln));
      if(n<256,
         {display(if(n<10,'  ',if(n<100, ' ', ' ')),n,'  ');
          kolom(0,n);
          rij(n+16)},
         display(eoln))};
   display('    ');
   hoofding(0);
   rij(32)}
:<function ascii>
```



Voorgedefinieerde controle functies (vervolg&voorbeeld) ...



```
ascii():
  {display('     ');
   for(n: 0,n:= n+1,n<16,display(if(n<10,'  ', ' '),n));
   display(eoln,eoln);
   for(m:32,m:=m+16,m<256,
       {display(if(m<10,'   ',if(m<100,'  ',' ')),m,'  ');
        for(n:0,n:=n+1,n<16,
            display(' ',char(m+n),' '));
        display(eoln)})}
:<function ascii>
```

transcript

Voorgedefinieerde controle

```
etiketten(breedte,kolommen,rijen):
  {lees(tekst):
     {display('geef ',tekst,': ');
      x: accept();
      display(eoln);
      x};
   naam: lees('naam');
   voornaam: lees('voornaam');
   adres: lees('adres');
   postnummer: lees('postnummer');
   gemeente: lees('gemeente');
   n1: breedte-size(voornaam)-size(naam)-1;
   n2: breedte-size(adres);
   n3: breedte-size(postnummer)-size(gemeente)-1;
   vulling(breedte*kolommen,'-');
   display(eoln);
   for(r:0,r:=r+1,r<rijen,
       {display(eoln,eoln);
```

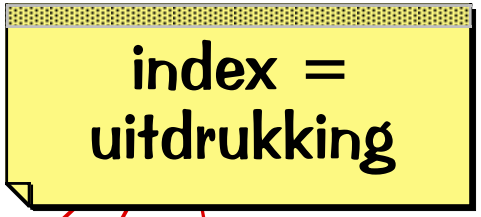
transcript

Voorgedefinieerde controle

```
    vulling(breedte*kolommen,'-');
    display(eoln);
    for(r:0,r:=r+1,r<rijen,
        {display(eoln,eoln);
         for(k:0,k:=k+1,k<kolommen,
             {display(voornaam,' ',naam);
              vulling(n1,' ')});
         display(eoln);
         for(k:0,k:=k+1,k<kolommen,
             {display(adres);
              vulling(n2,' ')});
         display(eoln);
         for(k:0,k:=k+1,k<kolommen,
             {display(postnummer,' ',gemeente);
              vulling(n3,' ')});
         display(eoln)});
    display(eoln,eoln);
    vulling(breedte*kolommen,'-');
    display(eoln)}
```

transcript

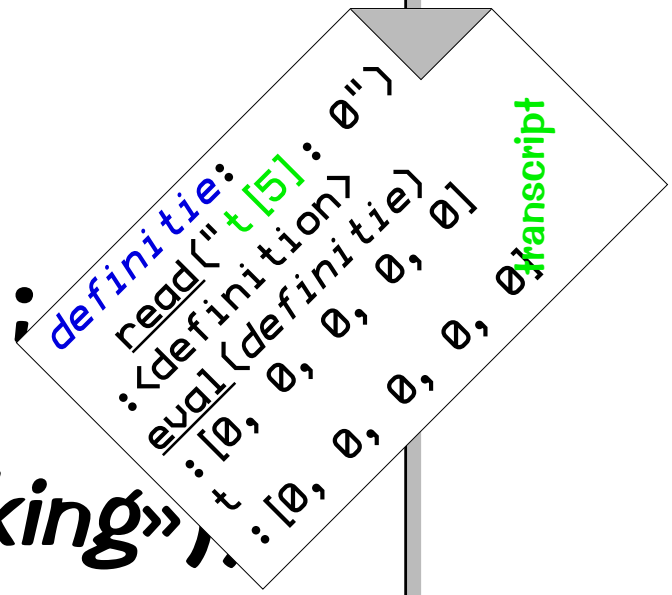
Gebruik van tabellen ...



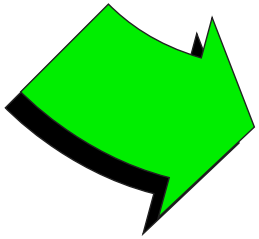
definitie ...	«*woord*[*index*]: *uitdrukking*»
indexering ...	«*woord*[*index*]»
wijziging ...	«*woord*[*index*]:= *uitdrukking*»

Definitie van tabellen ...

```
eval(«woord[index]: uitdrukking»)
                    ≡
{ siz: eval(«index»);
  tab[siz]: void;
  Ω:= add(«woord», tab, Ω);
  for(x:1,x:=x+1,not(x>siz),
      tab[x]:= eval(«uitdrukking»));
  tab}
```



dit heet de definitie van een tabel



Definitie van tabellen (#1) ...

siz: eval(«*index*»)

- «*index*» wordt geëvalueerd
- het resultaat wordt bewaard in siz
- indien siz geen natuurlijk getal is wordt een fout veroorzaakt

Definitie van tabellen (#2) ...

tab[siz]: void

- er wordt een tabel aangemaakt met lengte siz
- het resultaat wordt bewaard in tab

Definitie van tabellen (#3) ...

$\Omega :=$ add(«woord», tab, Ω)

tab wordt gebonden aan
«woord» binnen Ω

Definitie van tabellen (#4) ...

```
for(x:1,x:=x+1,not(x>siz),
    tab[x]:= eval(«uitdrukking»))
```

- de componenten van tab worden één voor één afgelopen met behulp van x
- elke componente wordt overschreven met een (nieuwe) waarde van «*uitdrukking*»

Definitie van tabellen (#5) ...

tab

tab wordt terruggegeven als waarde van de definitie

Definitie van tabellen (voorbeeld) ...

```
X[5]: 1
:[1, 1, 1, 1, 1]
X[3]: Y[4]: 0
:[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
n: 0
:0
T[10]: n:= n+1
:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
p: 0
:0
q: 0
:0
P[5]: Q[{q:=0; p:=p+1}]: q:= q+1
:[[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5]]
```

transcript

Indexering van tabellen ...

eval(«woord[*index*]»)

≡

{ tab: get(«woord»);
  tab[eval(«*index*»)]}

```
indexering:
  read("t[3]")
:⌜tabulation⌝
  eval(indexering)
:0
  t[3]
:1
```

transcript

de evaluatie van «woord[*index*]» betekent:

1) de binding van «woord» wordt bewaard in tab
2) indien tab geen tabel is wordt een fout veroorzaakt
3) de evaluatie van «*index*»
4) indien de waarde uit 3) geen positief getal is begrensd door de lengte van de tabel uit 2) wordt een fout veroorzaakt
5) de componente van tab met als index de waarde uit 3) wordt teruggegeven

dit heet de indexering van een tabel

Indexering van tabellen (voorbeeld)...

```
P[2]: 1
:[1, 1]
x: 0
:0
v: 0
:0
Q[3]: if(x<2,{s:v+P[x:=x+1]; v:=P[x]; s},1)
:[1, 2, 1]
x:= 0
:0
v:= 0
:0
R[4]: if(x<3,{s:v+Q[x:=x+1]; v:=Q[x]; s},1)
:[1, 3, 3, 1]
```

transcript

Indexering van tabellen (voorbeeld)...

```
Pascal(P,n):
   {x: 0;
    v: 0;
    Q[size(P)+1]: if(x<size(P),
                    {s: v+P[x:=x+1];
                     v:= P[x];
                     s},
                    1);
    if(n>2,Pascal(Q,n-1),Q)}
:<function Pascal>
Pascal(t[2]:1,5)
:[1, 5, 10, 10, 5, 1]
Pascal(t[2]:1,11)
:[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
```

transcript

Wijziging van tabellen ...

eval(«woord[*index*]:= *uitdrukking*»)

≡

```
{tab: get(«woord»);
 tab[eval(«index»)]:= eval(«uitdrukking»);
 tab}
```

de evaluatie van «woord[*index*]:= *uitdrukking*» betekent:

1) de binding van «woord» wordt bewaard in tab
2) indien tab geen tabel is wordt een fout veroorzaakt
3) de evaluatie van «*index*»
4) indien de waarde uit 3) geen positief getal is begrensd door de lengte van de tabel uit 2) wordt een fout veroorzaakt
5) de componente van tab met index de waarde uit 3) wordt vervangen door de waarde van «*uitdrukking*»
6) tab wordt teruggegeven

dit heet de wijziging van een tabel

transcript

```
wijziging:
 read("t[3]:= 1")
 :<assignment>
 eval(wijziging)
 :[0, 0, 1, 0, 0]
```

Wijziging van tabellen (voorbeeld 1) ...

```
Pascal(P,n):
   {Q[size(P)+1]: 1;
    for(x:2,x:=x+1,x<size(Q),Q[x]:=P[x-1]+P[x]);
    if(n>2,Pascal(Q,n-1),Q)}
:<function Pascal >
Pascal(t[2]: 1,11)
:[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
```

transcript

Wijziging van tabellen (voorbeeld 1 &vervolg) ...

```
Pascal(n):
   {P[n+1]: 1;
    for(r: 2,r:= r+1,not(r>n),
        for(k:r,k:=k-1,k>1,P[k]:=P[k]+P[k-1]));
    P}
:<function Pascal>
Pascal(5)
:[1, 5, 10, 10, 5, 1]
Pascal(11)
:[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
```

transcript

Wijziging van tabellen (voorbeeld 2) ...

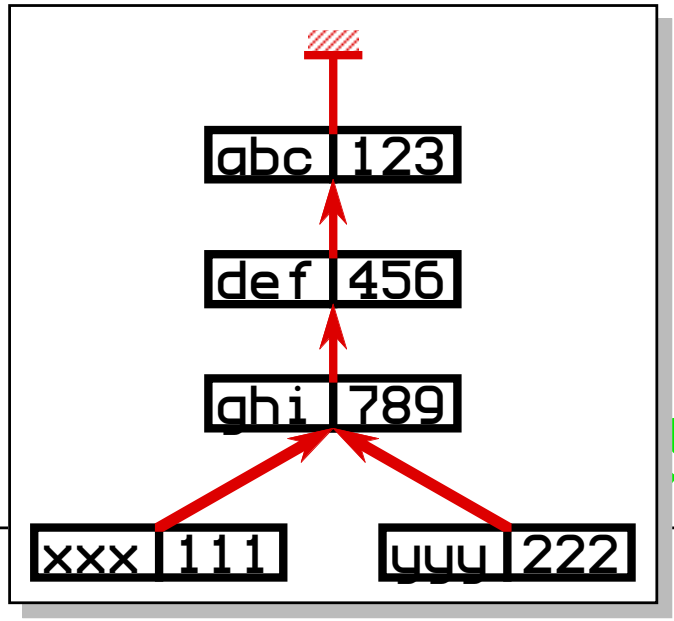
```
add(word,value,dict): [word,value,dict]
:<function add>
get(word,dict):
  if(is_void(dict),
     display('undefined identifier: ',word),
     if(dict[1]=word,dict[2],get(word,dict[3])))
:<function get>
set(word,value,dict):
  if(is_void(dict),
     display('undefined identifier: ',word),
     if(dict[1]=word,
        {dict[2]:= value;
         void},
        set(word,value,dict[3])))
:<function set>
```

transcript

zie vroeger

Wijziging van tabellen (voorbeeld O&-vervolg) ...

```
Omega: add('ghi',789,
           add('def',456,
               add('abc',123,void)))
:[ghi, 789, [def, 456, [abc, 123, <void>]]]
Omega1: add('xxx',111,Omega)
:[xxx, 111, [ghi, 789, [def, 456, [abc, 123, <void>]]]]
Omega:= add('yyy',222,Omega)
:[yyy, 222, [ghi, 789, [def, 456, [abc, 123, <void>]]]]
get('def',Omega)
:456
set('def',654,Omega1)
:<void>
get('def',Omega)
:654
get('xxx',Omega)
:undefined identifier: xxx
```



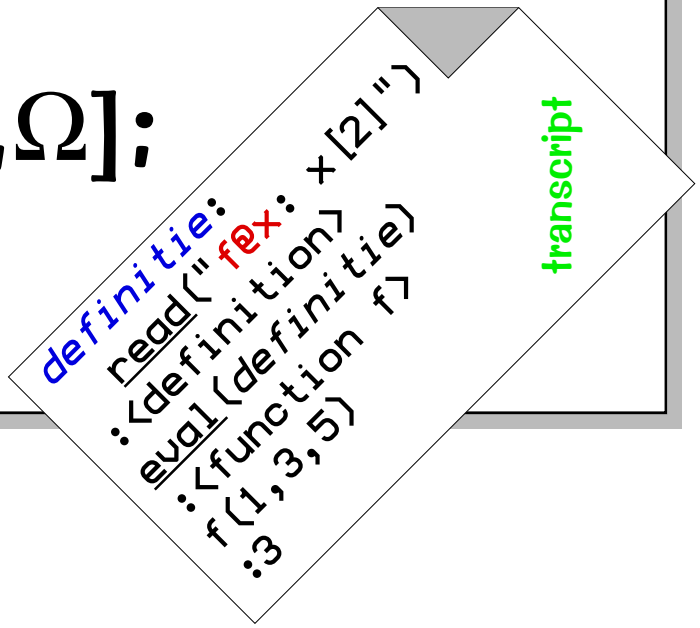
ranscript

Definitie van functies met tabellen ...

eval(«*woord*$_1$@*woord*$_2$: *uitdrukking*»)

≡

{Ω := add(«*woord*$_1$»,void,Ω):
fun: [«*woord*$_2$»,«*uitdrukking*»,Ω];
set(«*woord*$_1$»,fun,Ω):
fun}



Enkel deze stap is nieuw

Definitie van functies met tabellen (voorbeeld) ...

```
tab@x: x
:<function tab>
tab()
:[]
tab(1,2,3)
:[1, 2, 3]
tab(tab(1,2),tab(2,3),tab(3,4))
:[[1, 2], [2, 3], [3, 4]]
begin@x: x[size(x)]
:<function begin>
begin(display('hokus'),display('pokus'))
:hokuspokus
som@x:
 begin(s:0,for(n: 1,n:= n+1,not(n>size(x)),s:= s+x[n]),s)
:<function som>
som(1,2,3,4,5)
:15
```

transcript

Toepassing van functies met tabellen ...

eval(«*woord*$_1$@*woord*$_2$»)

≡

```
{ fun: get(«woord1»,Ω):
  dict: Ω;
  Ω:= bind(eval(«woord2»),fun[1],fun[3]);
  val: eval(fun[2]):
  Ω:= dict;
  val }
```

Enkel deze stap is nieuw

```
toepassing:
  read("fet")
  :<application>
  eval(toepassing)
  :1
```

transcript

Toepassing van functies met tabellen (vervolg) ...

$\Omega :=$ bind(eval(«*woord*$_2$»),fun[1],fun[3])

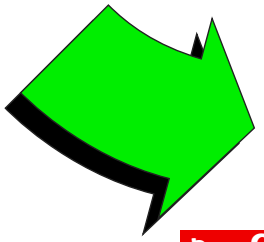
≡

$\Omega :=$ bind([val,...],[«*parameter*»,...],dict)

of

$\Omega :=$ bind([val,...],«*woord*»,dict)

bind maakt een binding tussen een tabel [val,...] en de parameters binnen het woordenboek van de functie en stelt Ω hieraan gelijk



Binding van parameters ...

bind([val,...],[«*parameter*»,...],dict)

≡

dict:= add(«*parameter*»,val,dict) ...

de binding van de tabel [val,...] aan de parameterlijst [«*parameter*»,...] betekent:

- indien ze niet dezelfde lengte hebben wordt een fout veroorzaakt
- elke val wordt gebonden aan de overeenstemmende «*parameter*» binnen dict

Binding van parameters (vervolg) ...

bind([val,...],«*woord*»,dict)
≡
add(«*woord*»,[val,...],dict)

de binding van de tabel [val,...] aan «*woord*» betekent dat de tabel wordt gebonden aan «*woord*» binnen dict

Binding van parameters (voorbeeld 1) ...

```
sorteer(p,q,r):
  if(p<q,if(q<r,display(p,q,r),
                if(p<r,display(p,r,q),display(r,p,q))),
        if(p<r,display(q,p,r),
                if(q<r,display(q,r,p),display(r,q,p))))
:<function sorteer>
sorteer(2,3,1)
:123
sorteer(3,1,2)
:123
z: tab(3,2,1)
:[3, 2, 1]
sorteer@z
:123
```

transcript

Binding van parameters (voorbeeld 2) ...

```
map@x:
   {op: x[1];
    tab[size(x)-1]: 0;
    for(n: 1,n:= n+1,n<size(x),tab[n]:= op(x[n+1]));
    tab}
:<function map>
pi:3.14159265358979
:3.14159
map(sin,0,pi/8,pi/4,3*pi/8,pi/2,5*pi/8,3*pi/4,7*pi/8,pi)
:[0.000000, 0.382683, 0.707107, 0.923880, 1.000000,
0.923880, 0.707107, 0.382683, 0.000000]
map(sqr(x): x*x,1,2,3,4,5)
:[1, 4, 9, 16, 25]
```

transcript

Voorgedefinieerde tabelfuncties ...

size(T): ...geef het aantal componenten van *T*...

geeft een fout indien het argument geen tabel of tekst voorstelt

Voorgedefinieerde tabelfuncties (vervolg) ...

reeds eerder informeel gedefinieerd

tab@T: T

begin@T: T[size(T)]

syntactische "suiker":
tab(a,b,...,z) ≡ [a,b,...,z]
begin(a,b,...,z) ≡ {a;b;...;z}

Voorgedefinieerde tabelfuncties (vervolg) ...

explode(T): ...maak een tabel met de letters uit *T*...

implode(L_1,L_2,...): ...maak tekst met de letters L_1,L_2,...

letters zijn tekst bestaande uit één teken

Voorgedefinieerde

```
PigLatin(tekst):
   {tab: explode(tekst);
    Copieer(n): if(and(n<size(tab), not(tab[n]=' ')),
                   {display(tab[n]);
                    Copieer(n+1)},
                   n);
    Klinker(k): or(k='a',or(k='e',or(k='i',or(k='o',k='u'))));
    Woord(n): if(and(n<size(tab),not(tab[n]=' ')),
                  {if(Klinker(tab[n]),
                      n:= Copieer(n),
                      {init: tab[n];
                       n:= Copieer(n+1);
                       display(init)});
                   display('ay ');
                   Woord(n+1)},
                  eoln);
    Woord(1)}
:<function PigLatin>
PigLatin('tea for two and two for tea')
:eatay orfay wotay anday wotay orfay etay
```

transcript

Voorgedefinieerde tabelfuncties (voorbeeld 2) ...

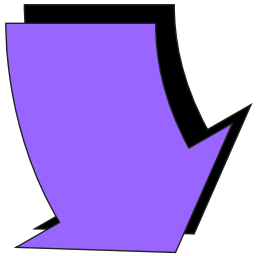
```
number_text(nbr,n):
  if(is_number(nbr),
     {txt: text(nbr);
      prf[n-size(txt)]: ' ';
      implode(prf)+txt},
     void)
:<function number_text>
number_text(1,10)
:          1
number_text(1234,10)
:       1234
number_text(99999999,10)
:  99999999
```

transcript

Gevalstudie: sorteren van een tabel

vb:

['do', 're', 'mi', 'fa', 'sol', 'la', 'si', 'do']



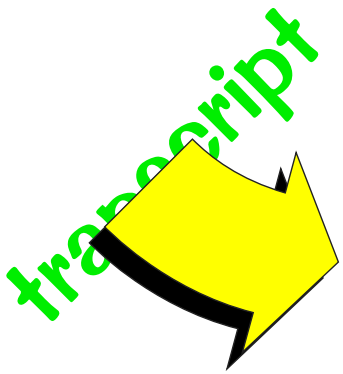
['do', 'do', 'fa', 'la', 'mi', 're', 'si', 'sol']

correcte keuze uit 8! mogelijkheden

```
Noten: ['do','re','mi','fa','sol','la','si','do']
:[do, re, mi, fa, sol, la, si, do]
eerste(T):
   {x: T[1];
    for(n:2,n:=n+1,not(n>size(T)),
        if(T[n]<x,x:=T[n],void));
    x}
:<function eerste>
eerste(Noten)
:do
verwissel(T,i,j):
   if(i=j,
      T,
      {x: T[i];
       T[i]:= T[j];
       T[j]:= x;
       T})
:<function verwissel>
verwissel(Noten,1,5)
:[sol, re, mi, fa, do, la, si, do]
```

we zoeken de waarde die op de eerste plaats moet komen

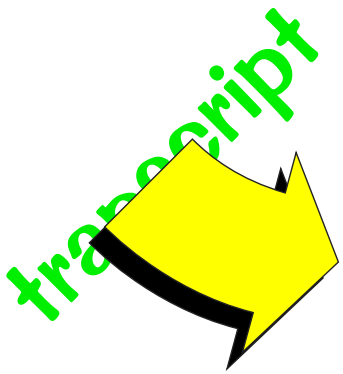
we verwisselen (indien nodig) de kleinste met de eerste componente



```
eersteVooraan(T):
   {x: T[1];
    for(n: 2,n:=n+1,not(n>size(T)),
        if(T[n]<x,
           {verwissel(T,1,n);
            x:= T[n]},
           void));
    T}
:<function eersteVooraan>
eersteVooraan(Noten)
:[do, re, mi, fa, sol, la, si, do]
TweedeOpDeTweedePlaats(T):
   {x: T[2];
    for(n:3,n:=n+1,not(n>size(T)),
        if(T[n]<x,
           {verwissel(T,2,n);
            x:= T[n]},
            void));
    T}
:<function TweedeOpDeTweedePlaats>
```

we combineren de twee...

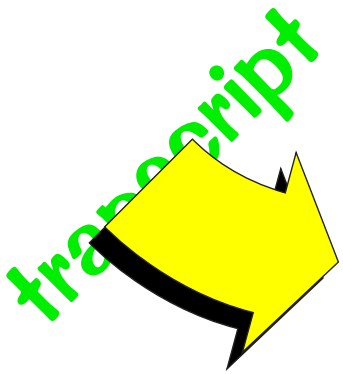
we doen hetzelfde met de tweede



```
TweedeOpDeTweedePlaats(Noten)
:[do, do, re, mi, sol, fa, si, la]
OpDeGoedePlaats(T,k):
   {x: T[k];
    for(n:k+1,n:=n+1,not(n>size(T)),
        if(T[n]<x,
           {verwissel(T,k,n);
            x:=T[k]},
           void));
    T}
:<function OpDeGoedePlaats>
OpDeGoedePlaats(Noten,3)
:[do, do, fa, re, sol, mi, si, la]
OpDeGoedePlaats(Noten,4)
:[do, do, fa, la, sol, re, si, mi]
OpDeGoedePlaats(Noten,5)
:[do, do, fa, la, mi, sol, si, re]
OpDeGoedePlaats(Noten,6)
:[do, do, fa, la, mi, re, sol, si]
OpDeGoedePlaats(Noten,7)
:[do, do, fa, la, mi, re, si, sol]
```

we maken een functie om de k^e waarde op de k^e plaats te krijgen...

...en die passen we toe to de tabel volledig geordend is



Gevalstudie: sorteren van een tabel (vervolg)

```
Noten: ['do','re','mi','fa','sol','la','si','do']
:[do, re, mi, fa, sol, la, si, do]
Sorteren(T):
   for(k:1,k:=k+1,k<size(T),
       {x: T[k];
        for(n:k+1,n:=n+1,not(n>size(T)),
            if(T[n]<x,
               {T[k]:= T[n];
                T[n]:= x;
                x:=T[k]},
               void));
        T})
:<function Sorteren>
Sorteren(Noten)
:[do, do, fa, la, mi, re, si, sol]
```

tenslotte voegen we alles samen in één functie

transcript

Synthese van de techniek:

$$\text{sort}(D) = \begin{cases} D & \text{als } \#D \leq 1 \\ \{m\} \oplus \text{sort}(D \setminus \{m\}) & \text{als } \#D > 1 \end{cases}$$

$m = \min D$

Performantie van de techniek:

- bepalen van m=min D vraagt #D stappen
- uitgaande van de oorspronkelijke D zijn #D iteraties nodig

$$\#\text{stappen} = \#D + (\#D-1) + (\#D-2) + \ldots + 2 + 1$$

$$= \#D.(\#D+1)/2$$

voorbeeld: 1000 elementen sorteren
vraagt een half miljoen stappen

Verbetering van de techniek:

$$\text{sort}(D) = \begin{cases} D & \text{als } \#D \leq 1 \\ \text{sort}(D_1) \oplus \text{sort}(D_2) & \text{als } \#D > 1 \end{cases}$$

$$D_1 \cap D_2 = \varnothing \text{ en } D_1 \cup D_2 = D$$

$$\max D_1 \leq \min D_2$$

```
V: [543,1085,390,932,779,237,84,626,1168,473]
:[543, 1085, 390, 932, 779, 237, 84, 626, 1168, 473]
L: 1
:1
R: size(V)
:10
M: V[(L+R)//2]
:779
while(V[L]<M,L:=L+1)
:2
while(V[R]>M,R:=R-1)
:<void>
{x:V[L];V[L]:=V[R];V[R]:=x;R:=R-1;L:=L+1;V}
:[543, 473, 390, 932, 779, 237, 84, 626, 1168, 1085]
while(V[L]<M,L:=L+1)
:4
while(V[R]>M,R:=R-1)
:8
{x:=V[L];V[L]:=V[R];V[R]:=x;R:=R-1;L:=L+1;V}
:[543, 473, 390, 626, 779, 237, 84, 932, 1168, 1085]
```

zoek een spilwaarde

zoek vanaf links de eerste≥ spil

zoek vanaf rechts de eerste≤ spil

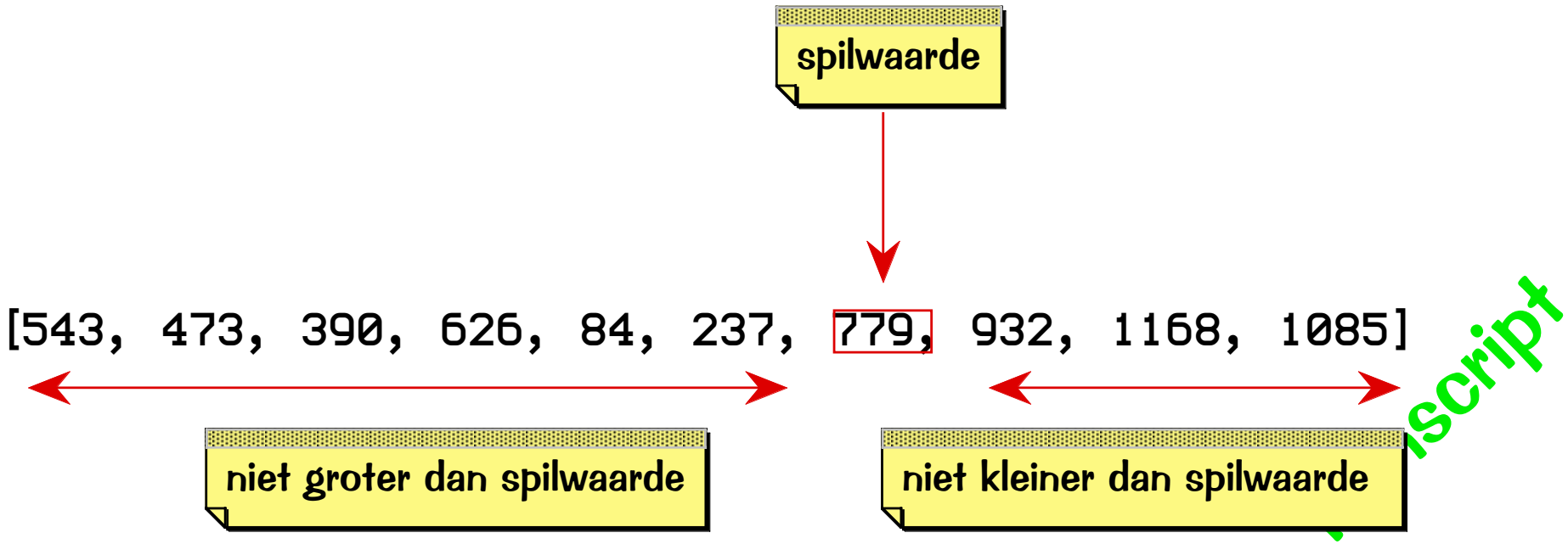
dit stopt vanzelf!

verwissel deze 2

en ga gewoon verder

transcript

```
while(V[L]<M,L:=L+1)
:<void>
while(V[R]>M,R:=R-1)
:<void>
{x:=V[L];V[L]:=V[R];V[R]:=x;R:=R-1;L:=L+1;V}
:[543, 473, 390, 626, 84, 237, 779, 932, 1168, 1085]
while(V[L]<M,L:=L+1)
:7
while(V[R]>M,R:=R-1)
:<void>
L
:7
R
:6
```



```
partitie(V,L,R):
   {M: V[(L+R)//2];
    x: void;
    while(not(L>R),
          {while(V[L]<M,L:=L+1);
           while(V[R]>M,R:=R-1);
           if(not(L>R),
              {x:=V[L];
               V[L]:=V[R];
               V[R]:=x;
               L:=L+1;
               R:=R-1},
              void)});
    display('L = ', L, ' R = ', R)}
:<function partitie>
V: [543,1085,390,932,779,237,84,626,1168,473]
:[543, 1085, 390, 932, 779, 237, 84, 626, 1168, 473]
partitie(V,1,10)
:L = 7 R = 6
V
:[543, 473, 390, 626, 84, 237, 779, 932, 1168, 1085]
```

we steken heel dit proces in één functie

dit is de proef op de som

transcript

Verbetering van de techniek (vervolg) :

```
partitie(V,1,6)
:L = 4 R = 2
partitie(V,1,2)
:L = 2 R = 1
partitie(V,4,6)
:L = 5 R = 4
partitie(V,5,6)
:L = 6 R = 5
partitie(V,7,10)
:L = 9 R = 7
partitie(V,6,7)
:L = 7 R = 5
partitie(V,9,10)
:L = 10 R = 9
V
:[84, 237, 390, 473, 543, 626, 779, 932, 1085, 1168]
```

hier kunnen we nu héél de tabel mee ordenen

transcript

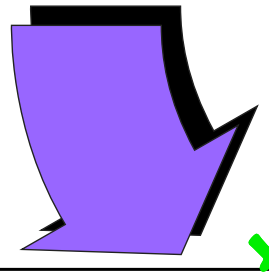
```
sort(V,LL,RR):
   if(LL<RR,
       {L: LL;
        R: RR;
        p: V[(L+R)//2];
        x: void;
        while(not(L>R),
              {while(V[L]<p,L:=L+1);
               while(V[R]>p,R:=R-1);
               if(not(L>R),
                  {x:=V[L];
                   V[L]:=V[R];
                   V[R]:=x;
                   L:=L+1;
                   R:=R-1},
                 void)});
        sort(V, LL, R);
        sort(V, L, RR)},
      V)
:<function sort>
```

we steken heel dit proces weer in één functie

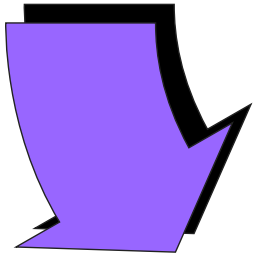
transcript

```
V:  [543,1085,390,932,779,237,84,626,1168,473]
:[543, 1085, 390, 932, 779, 237, 84, 626, 1168, 473]
sort(V,1,10)
:[84, 237, 390, 473, 543, 626, 779, 932, 1085, 1168]
y:1
:1
V[100]:y:=(y+4253)\\1237
:[543, 1085, 390, 932, 237, 779, 84, 626, 1168, 473, 1015,
320, 862, 167, 709, 14, 556, 1098, 403, 945, 250, 792, 97,
639, 1181, 486, 1028, 333, 875, 180, 722, 27, 569, 1111,
416, 958, 263, 805, 110, 652, 1194, 499, 1041, 346, 888,
193, 735, 40, 582, 1124, 429, 971, 276, 818, 123, 665,
1207, 512, 1054, 359, 901, 206, 748, 53, 595, 1137, 442,
984, 289, 831, 136, 678, 1220, 525, 1067, 372, 914, 219,
761, 66, 608, 1150, 455, 997, 302, 844, 149, 691, 1233,
538, 1080, 385, 927, 232, 774, 79, 621, 1163, 468, 1010]
```

we proberen de functie uit op de oorspronkelijke tabel, en we proberen ook eens een grotere tabel



transcript



```
sort(V,1,100)
:[14, 27, 40, 53, 66, 79, 84, 97, 110, 123, 136, 149, 167,
180, 193, 206, 219, 232, 237, 250, 263, 276, 289, 302, 320,
333, 346, 359, 372, 385, 390, 403, 416, 429, 442, 455, 468,
473, 486, 499, 512, 525, 538, 543, 556, 569, 582, 595, 608,
621, 626, 639, 652, 665, 678, 691, 709, 722, 735, 748, 761,
774, 779, 792, 805, 818, 831, 844, 862, 875, 888, 901, 914,
927, 932, 945, 958, 971, 984, 997, 1010, 1015, 1028, 1041,
1054, 1067, 1080, 1085, 1098, 1111, 1124, 1137, 1150, 1163,
1168, 1181, 1194, 1207, 1220, 1233]
```

transcript

Performantie van de verbeterde techniek:

- de uitvoering van de partitie op D vraagt #D stappen
- uitgaande van de oorspronkelijke D zijn $1+2+4+\ldots+2^k$ toepassingen van sort nodig

$$1.\#D + 2.(\#D/2) + 4.(\#D/4) + \ldots + 2^k.(\#D/2^k) = \#D.k = \#D.\log_2(\#D)$$

met $k = \log_2(\#D)$

in de veronderstelling dat de partitie telkens twee gelijke delen oplevert

voorbeeld: 1000 elementen sorteren vraagt 10000 stappen!

Binding van parameters opnieuw

Dit is hetzelfde geval als eerder besproken

bind(«*argument*»,«*woord*»,dict)

≡

dict:= add(«*woord*»,eval(«argument»),dict)

de binding van «*argument*» aan «*woord*» betekent:

1) de evaluatie van «*argument*»
2) de binding van «*woord*» en de waarde uit 1) in dict

Binding van parameters opnieuw (vervolg)

bind(«*argument*»,«*woord(parameter,...)*»,dict)

≡

{ fun: [[«*parameter*»,...],«*argument*»,Ω];
  dict:= add(«*woord*»,fun,dict)}

het binden van een argument aan een parameter van de vorm «*woord(parameter,...)*» betekent:

1) «argument» wordt *niet* geëvalueerd
2) een functie waarde wordt aangemaakt aan de hand van [«*parameter*»,...], «argument» en Ω
3) de waarde van 2) wordt gebonden aan «woord» binnen dict

Binding van parameters opnieuw (vervolg)

bind(«*argument*»,«*woord*$_1$*@woord*$_2$»,dict)

≡

{ fun: [«*woord*$_2$»,«*argument*»,Ω]:
  dict:= add(«*woord*$_1$»,fun,dict)}

het binden van een argument aan een parameter van de vorm «*woord*$_1$*@woord*$_2$» betekent:

1) «argument» wordt *niet* geëvalueerd
2) een functie waarde wordt aangemaakt aan de hand van «*woord*$_2$», «argument» en Ω
3) de waarde van 2) wordt gebonden aan «woord» binnen dict

Binding van parameters opnieuw (vervolg) ...

```
f(g(x)): g(1)+g(2)+g(3)+g(4)
:<function f>
f(x*x)
:30
collect(filter(x),t):
   {sum: 0;
    for(i:1,i:=i+1,not(i>size(t)),sum:=sum+filter(t[i]));
    sum}
:<function collect>
data: [1,2,3,5,7,11,13,17,19]
:[1, 2, 3, 5, 7, 11, 13, 17, 19]
collect(x*x,data)
:1028
```

g(x): x*x

filter(x): x*x

transcript

Voorgedefinieerde logische functies (opnieuw) ...

true(x(),y()): x()

false(x(),y()): y()

and(p,q()): p(q(),false)

or(p,q()): p(true,q())

not(p): p(false,true)

formele definitie

Voorgedefinieerde controle (vervolg) ...

```
true(p(),q()): p()
:<function true>
false(p(),q()): q()
:<function false>
true(display('T'),display('F'))
:T
and(p,q()): p(q(),false)
:<function and>
or(p,q()): p(true,q())
:<function or>
not(p): p(false,true)
:<function not>
and(true,false)
:<function false>
or(false,true)
:<function true>
not(false)
:<function true>
```

transcript

Voorgedefinieerde controle functies (vervolg) ...

if(cond,then(),else()): cond(then(),else())

formele definitie

Voorgedefinieerde controle functies (vervolg) ...

```
if(cond,then(),else()):
cond(then(),else())
:<function if>
if(true,display('T'),display('F'))
:T
if(1>2,display('T'),display('F'))
:F
1>2
:<native function false>
```

transcript

Voorgedefinieerde controle functies (vervolg) ...

```
while(cond(),expression()):
  {loop(value,pred):
     pred(loop(expression(),cond()),value);
    loop(void,cond())}
```

formele definitie

Voorgedefinieerde controle functies (vervolg) ...

```
while(cond(),expression()):
  {loop(value,pred):
     pred(loop(expression(),cond()),value);
   loop(void,cond())}
:<function while>
i: 0
:0
while(i<10,display(i:=i+1))
:12345678910
p: 33
:33
q: 121
:121
while(not(p=q),if(p>q,p:=p-q,q:=q-p))
:11
```

transcript

Voorgedefinieerde controle functies (vervolg) ...

```
until(cond(),expression()):
  { loop(value,pred):
      pred(value,loop(expression(),cond())):
    loop(expression(),cond())}
```

formele definitie

Voorgedefinieerde controle functies (vervolg) ...

```
for(init,incr(),cond(),expression()):
    {value: void;
      while(cond(),
        {value:= expression();
          incr();
          value})}
```

formele definitie

Binding van parameters opnieuw (vervolg) ...

```
merge(op(x,y),left,right):
   {s: size(left);
    x: 0;
    if(s=size(right),
       result[s]: op(left[x:=x+1],right[x]),
       void)}
:<function merge>
merge(if(x>y,x,y),[1,4,2,5,7,8,3],
                  [2,3,3,6,5,1,9])
:[2, 4, 3, 6, 7, 8, 9]
```

transcript

Voorgedefinieerde type functies ...

is_void(U):	*U* is void
is_number(U):	*U* is een getal
is_fraction(U):	*U* is een breuk
is_text(U):	*U* is tekst
is_table(U):	*U* is een tabel
is_function(U):	*U* is een functie

```
p++q:
  if(or(is_number(p),is_fraction(p)),
     if(or(is_number(q),is_fraction(q)),
        p+q,
        if(is_table(q),
           {i: 0;
            r[size(q)]: p++q[i:=i+1]},
           void)),
     if(is_table(p),
        if(or(is_number(q),is_fraction(q)),
           {i: 0;
            r[size(p)]: p[i:=i+1]++q},
           if(is_table(q),
              {i: 0;
               n: if(size(p)>size(q),size(p),size(q));
               r[n]: if((i:=i+1)>size(p),
                        q[i],
                        if(i>size(q),
                           p[i],
                           p[i]++q[i]))},
              void)),
        void))
```

Voorgedefinieerde type functies (vervolg) ...

```
:<function ++>
1++2
:3
1++[1,2,3]
:[2, 3, 4]
[4,5]++3.5
:[7.5, 8.5]
[1,2,3]++[4,5]
:[5, 7, 3]
[[1],1.5]++[1,[2,3],0.5]
:[[2], [3.5, 4.5], 0.5]
'abc'++123
:<void>
```

transcript