1 Inleiding

- processen 2
- waarden 12
- variabelen 15
- functies 19
- voorbeelden 42

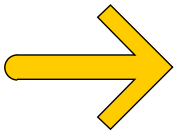
taal

Informatica = de studie van complexe processen aan de hand van uitvoerbare modellen

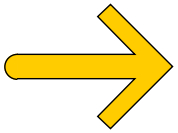
abstractie

computers

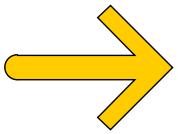
Processen...



hebben een gedrag ...

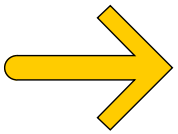


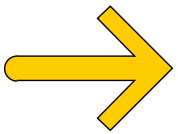
kennen tijd ...

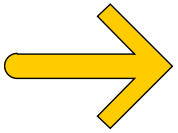


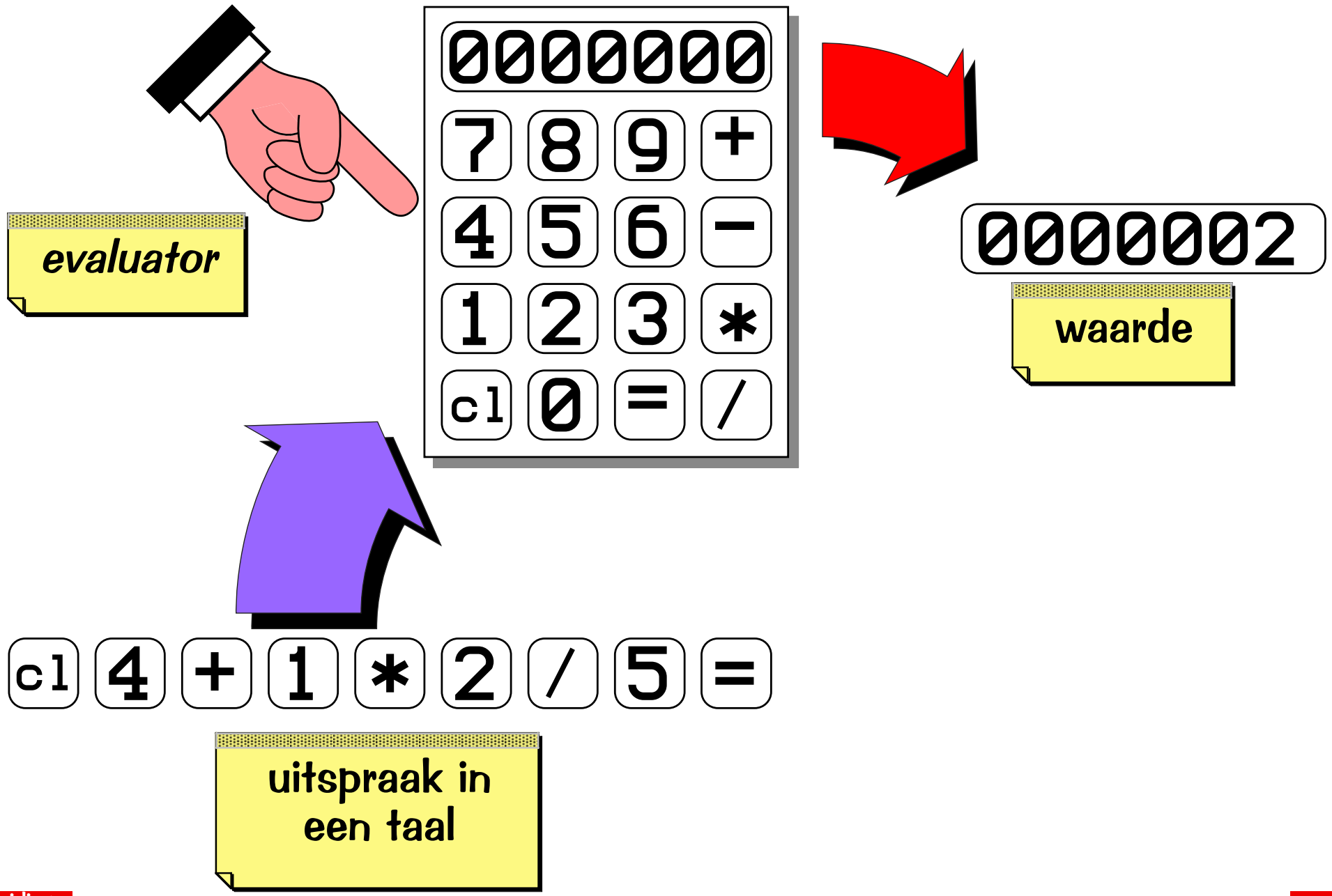
worden beschreven
met een taal ...

Uitvoerbare processen...

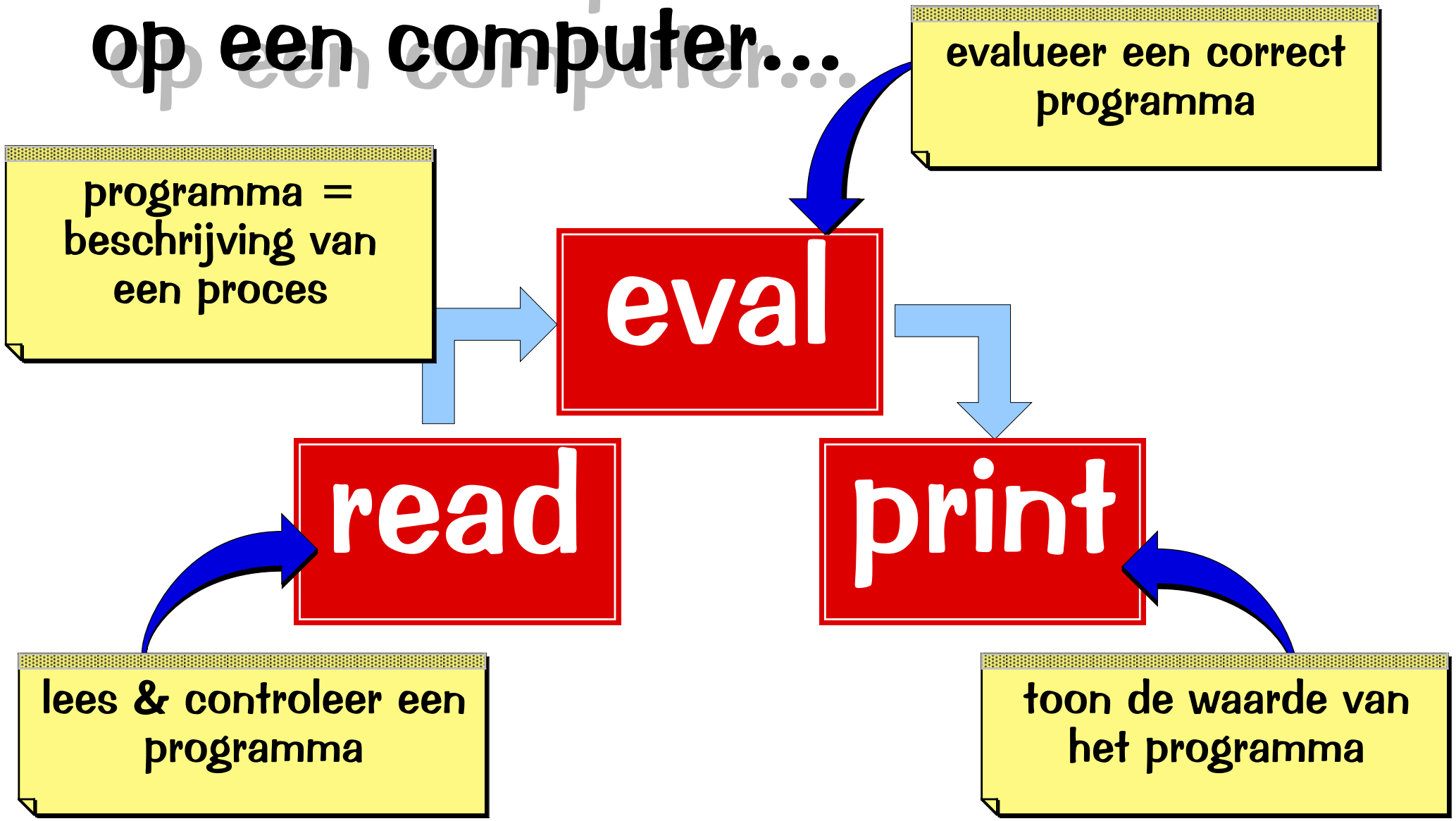

worden beschreven in een taal ...


vereisen een evaluator ...


produceren een waarde ...


0000000
7 8 9 +
4 5 6 −
1 2 3 *
cl 0 = /
evaluator
0000002
waarde
cl 4 + 1 * 2 / 5 =
uitspraak in
een taal

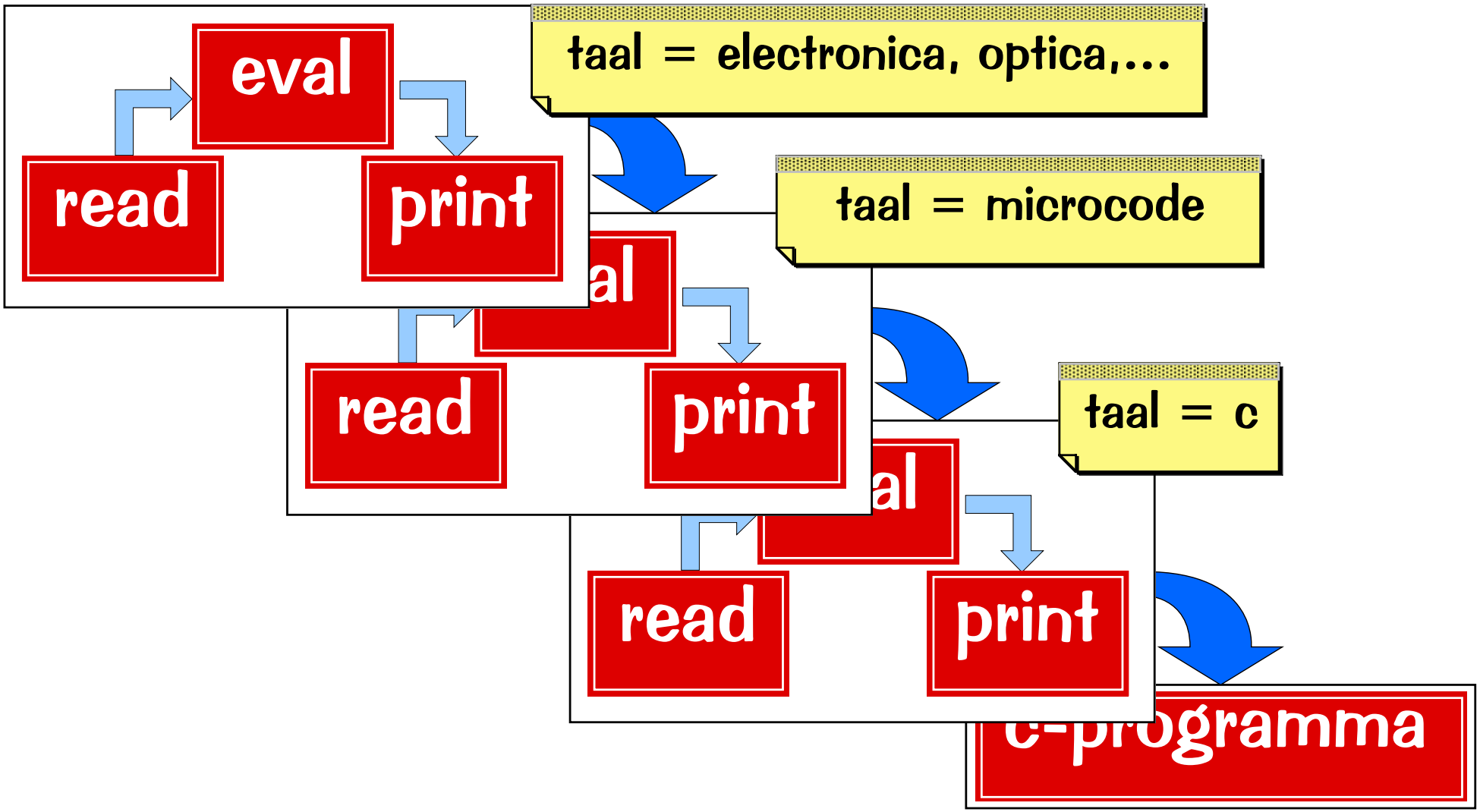
Uitvoerbaar proces op een computer...



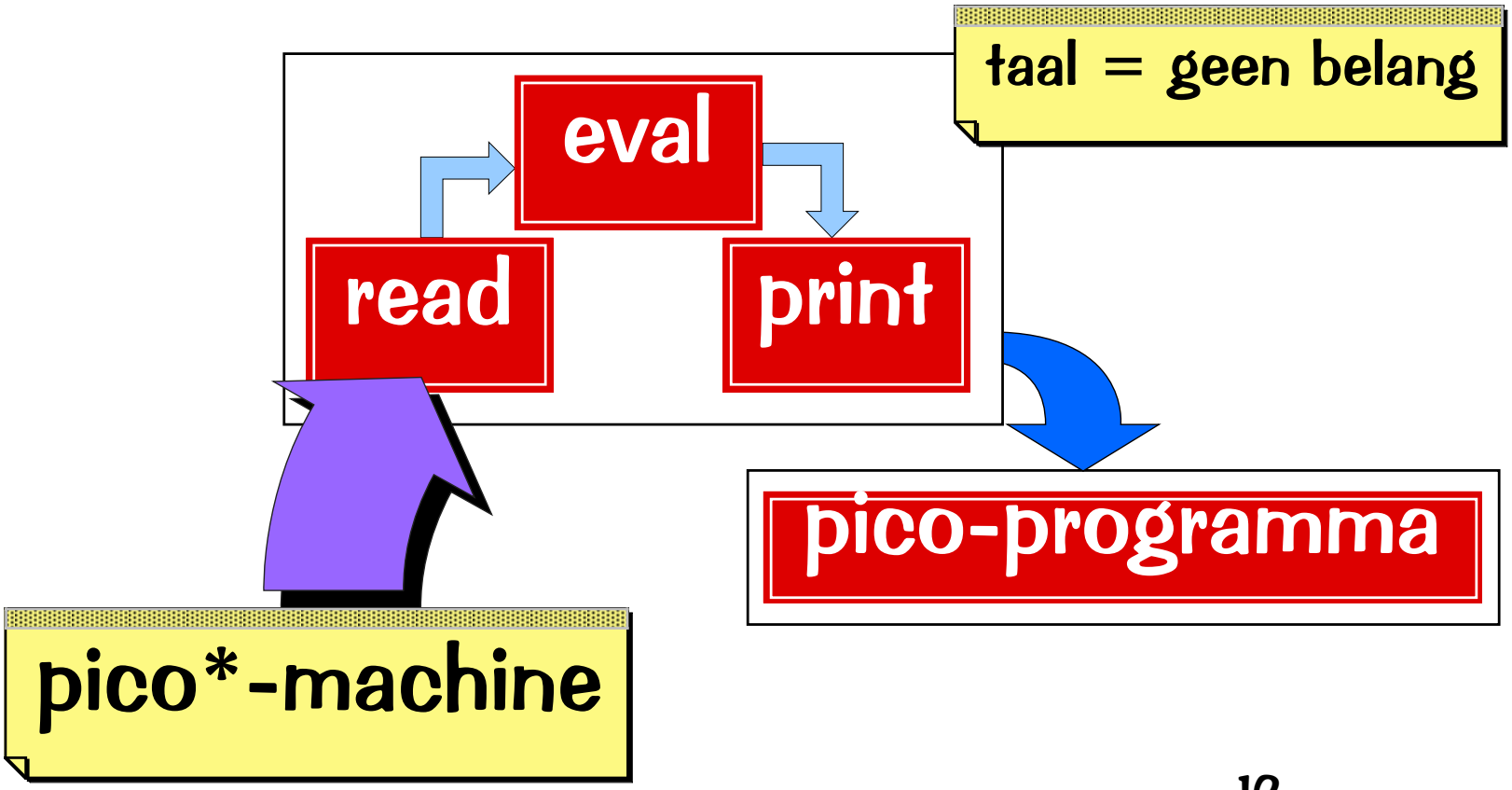
Een computer ...

- ... is een proces dat processen evalueert
- ... wordt beschreven door een programma
- ... bouwen = een programma schrijven

bij een pc is dit...

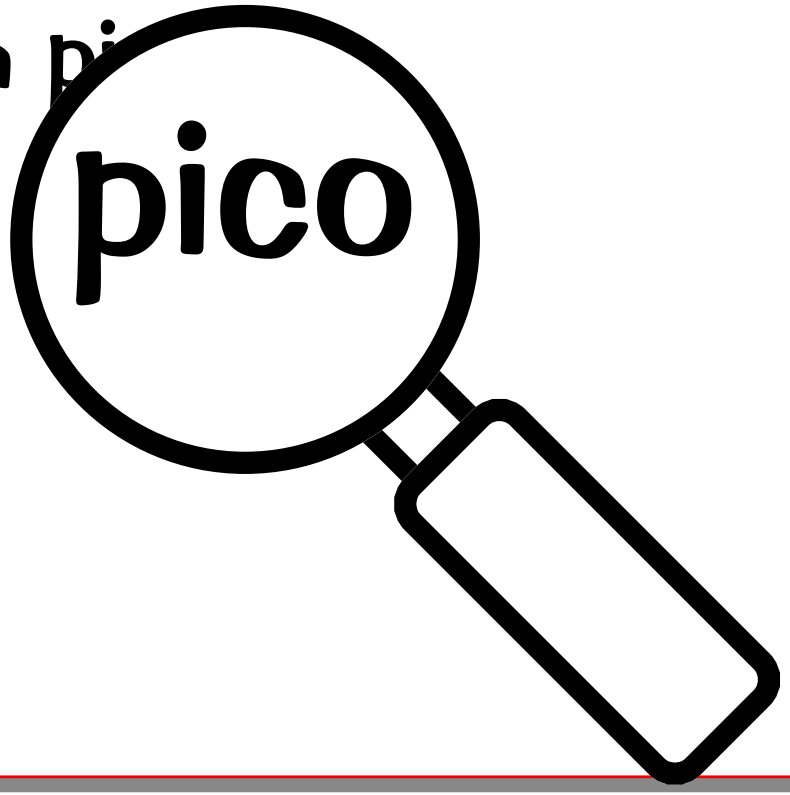


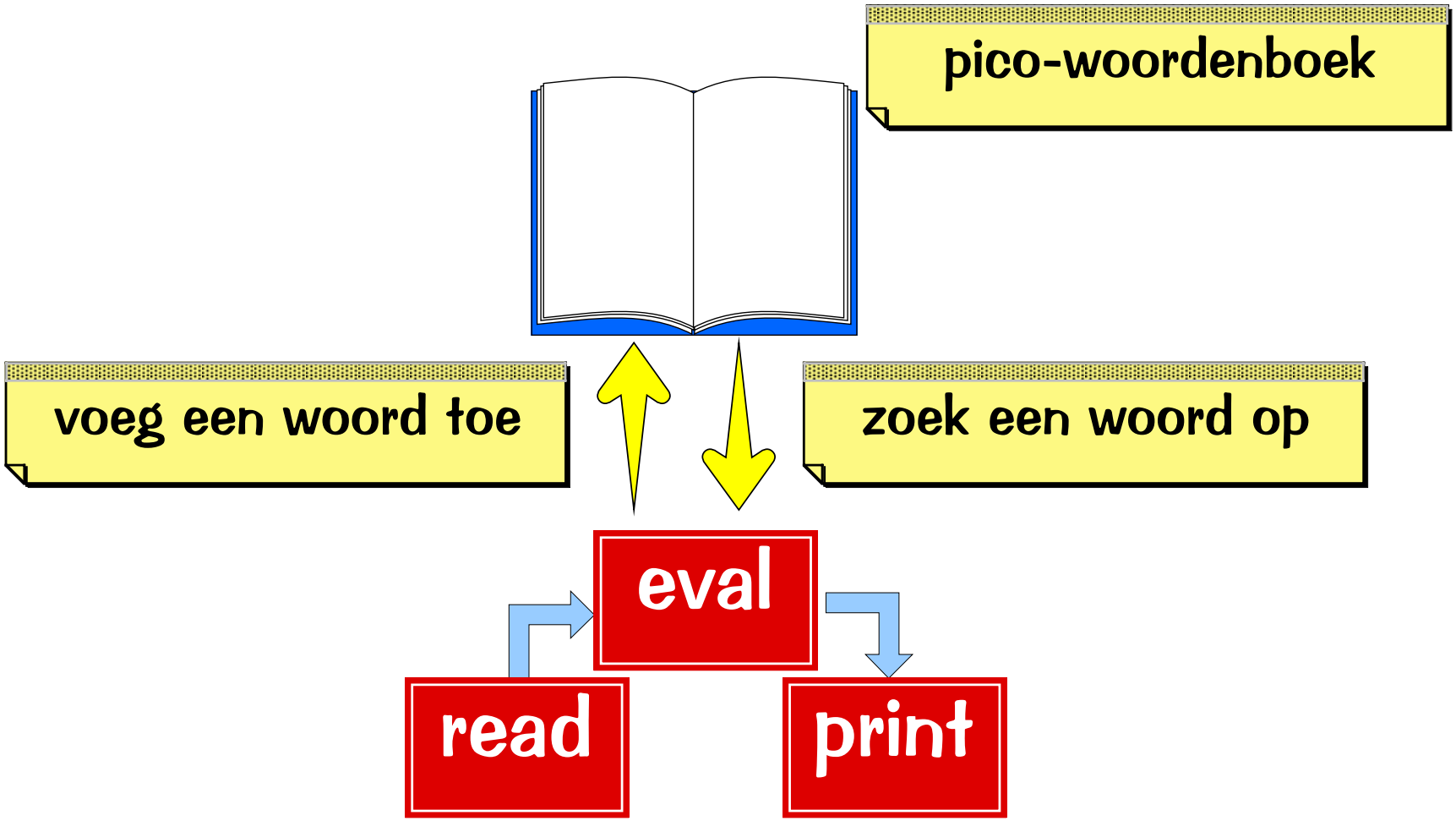
Eenvoudig model van een computer...



*pico = 10 $^{-12}$ = zéér klein

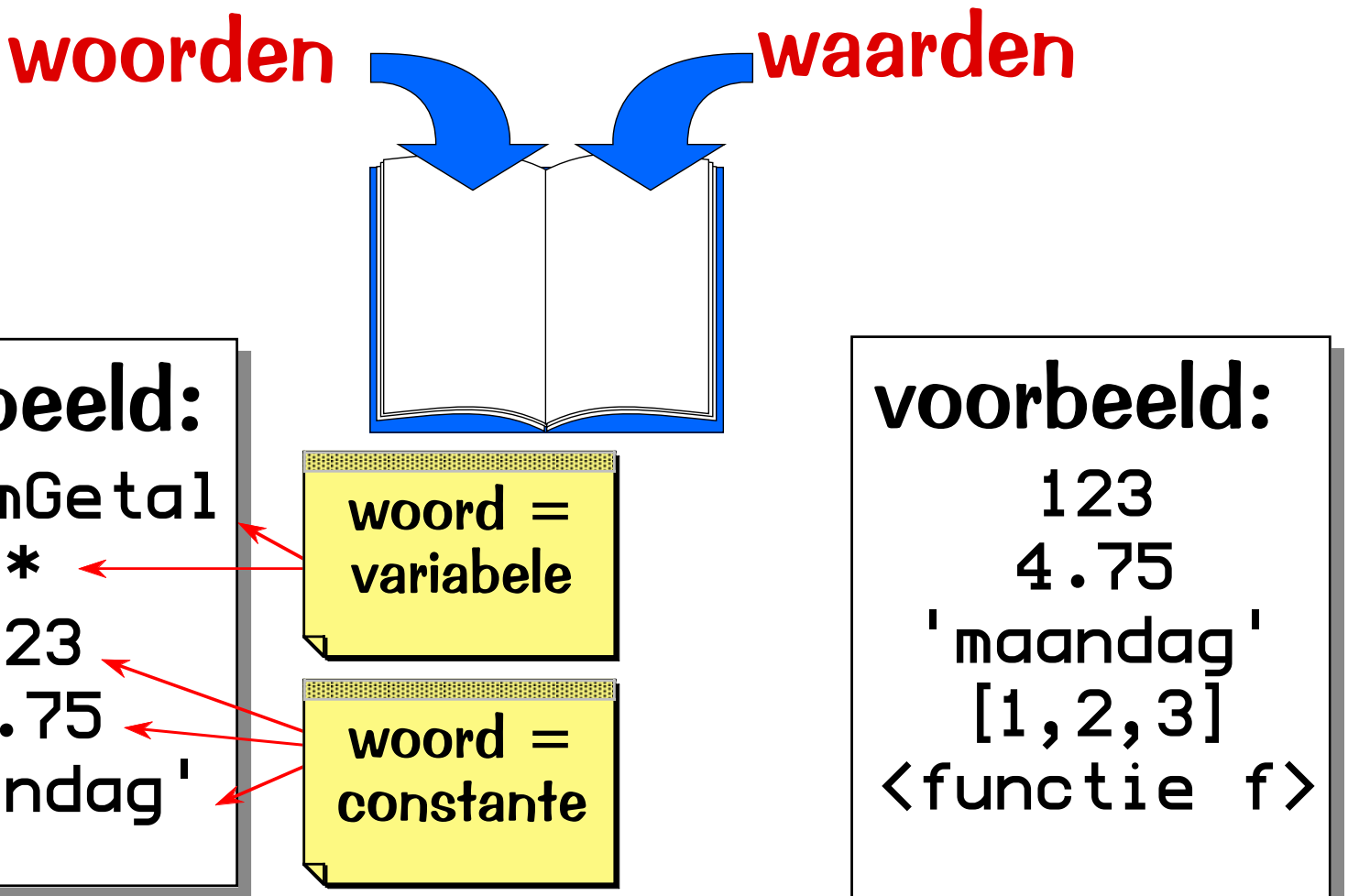
- de pico-machine evalueert pico-processen
- pico-processen worden beschreven in pi





de pico-evaluator beheert een woordenboek waarin woorden verbonden worden met waarden

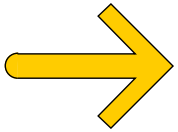
Structuur van het woordenboek...



- Bij aanvang is het woordenboek opgevuld met *constanten* = namen waarvan de waarde niet veranderd kan worden
- Er zijn *numerieke* en *tekstuele* constanten

naam = 123
naam = 12.345

naam = 'abracadabra'



Het woordenboek kan onbeperkt worden uitgebreid met *variabelen* = woorden waarvan de waarde kan overschreven worden

aanmaken van een nieuwe naam:

naam : *uitdrukking*

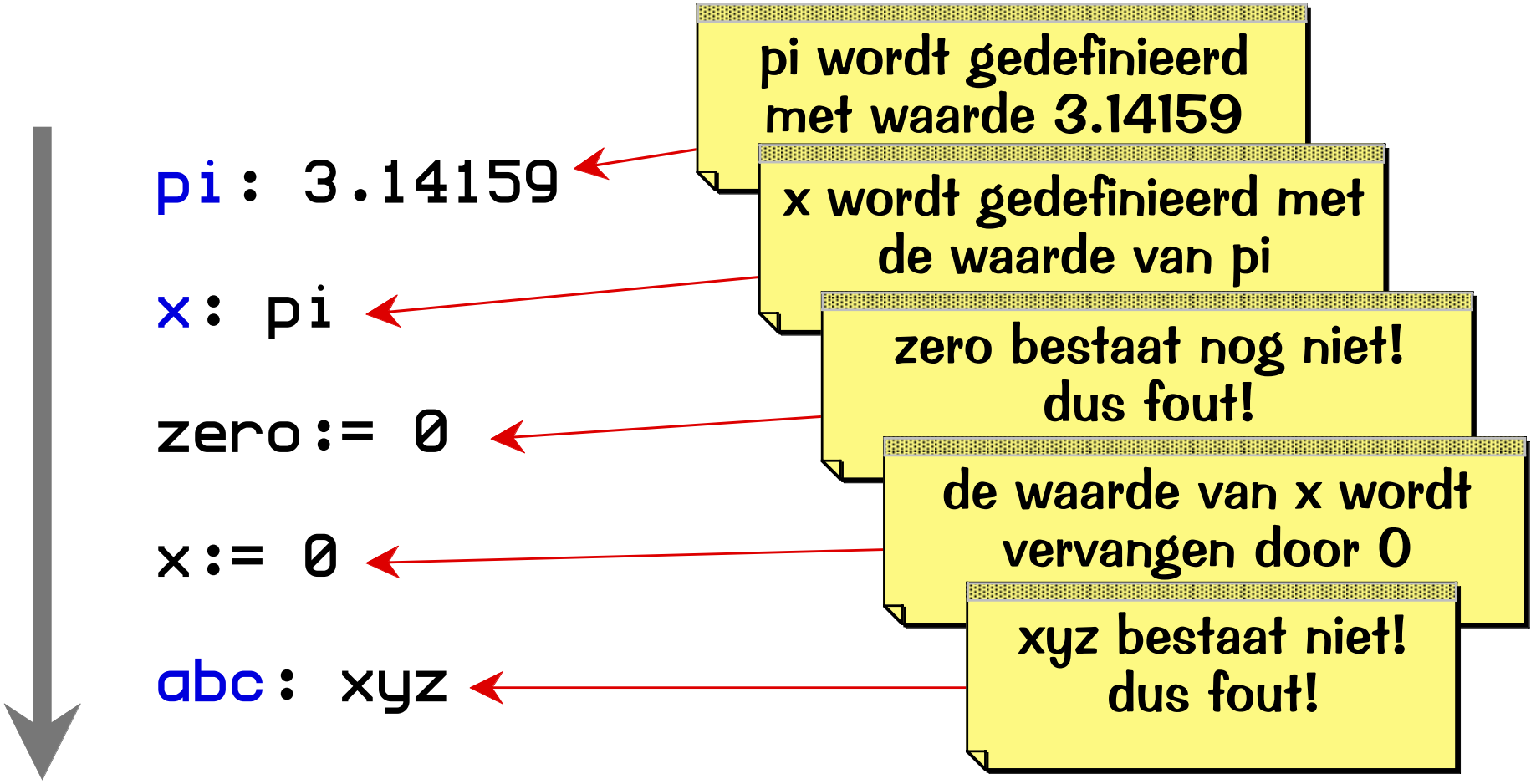
- de *naam* wordt toegevoegd achteraan het woordenboek
- de *uitdrukking* moet een geldige pico uitspraak zijn
- de waarde van de *uitdrukking* wordt toegekend aan de naam

de waarde van een naam wijzigen:

naam := *uitdrukking*

- de *naam* moet bestaan in het woordenboek
- de *uitdrukking* is een geldige pico uitspraak
- de waarde van de *uitdrukking* wordt toegekend aan de *naam*

voorbeelden:



tabellen en functies:

Naast numerieke en tekstuele waarden bestaan er ook *tabellen* en *functies*

- *tekstuele waarden* = waarden van tekstuele constanten
- *numerieke waarden* = waarden van numerieke constanten
- *tabellen*: zie later
- *functies*: wordt nu behandeld

functie = programmafragment

naam(*uitdrukking*$_1$, *uitdrukking*$_2$, ...)

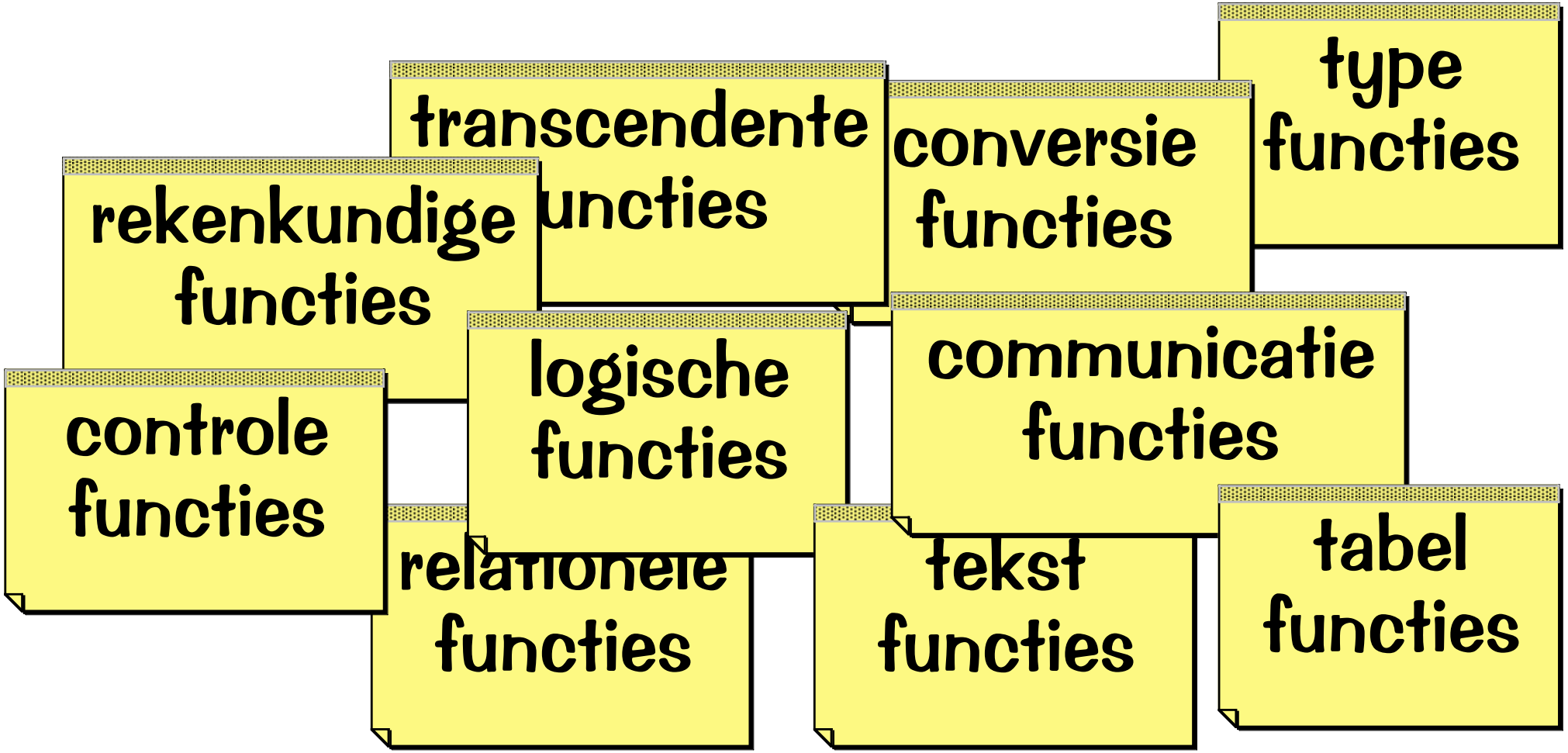
- functies zijn gebonden aan een *naam*
- functies kunnen worden *opgeroepen*
- bij oproep bepaalt de waarde verbonden met de naam *wat* er gebeurt
- *uitdrukking*$_1$, *uitdrukking*$_2$, ... bepalen waarop de functie van toepassing is

operatoren:

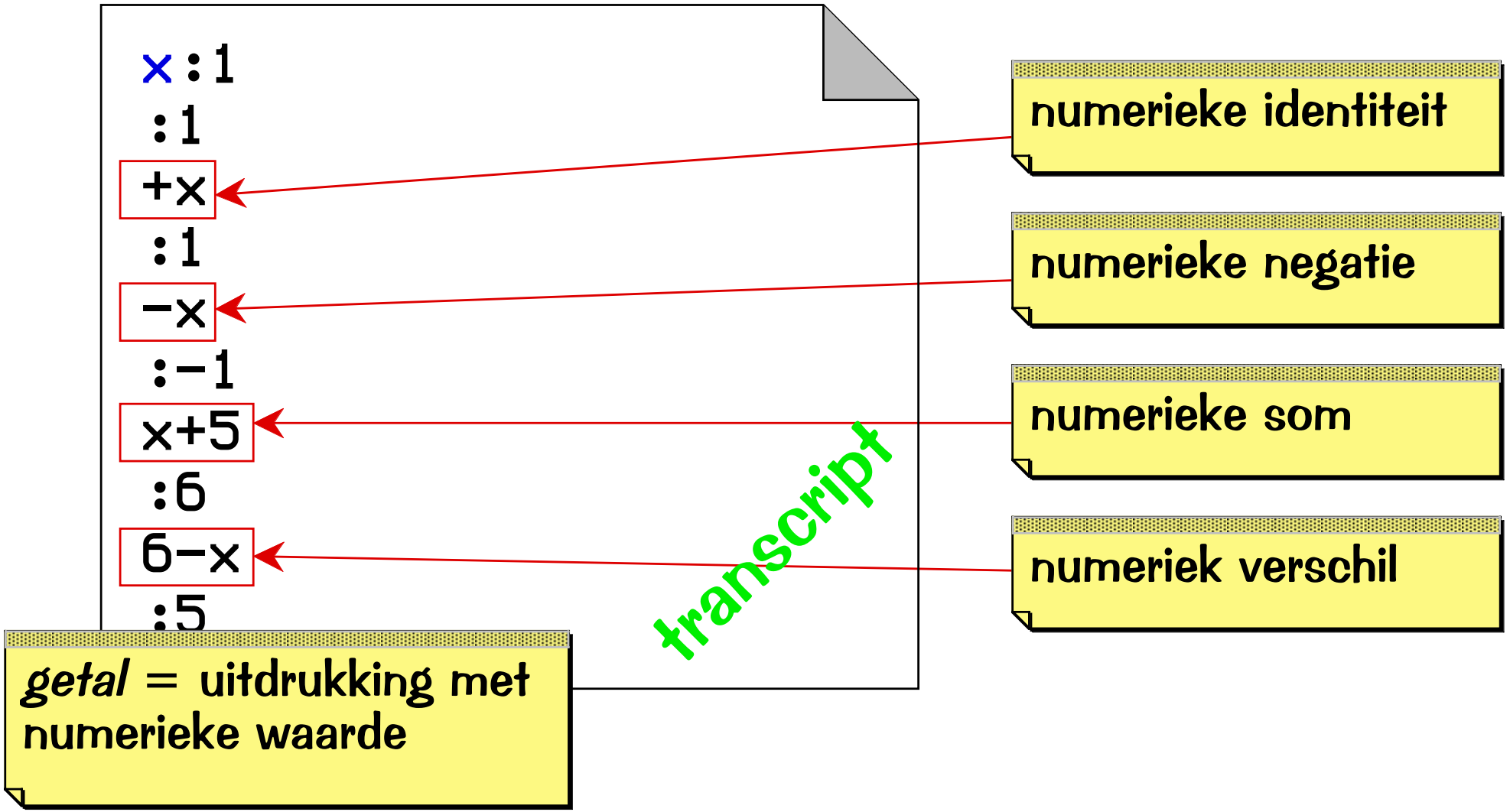
Indien de functienaam een *operator* is, en het aantal uitdrukkingen één of twee is, mag rekenkundige notatie gebruikt worden

`+(1,2)`	⇔	`1+2`	binaire notatie
`-(3.45)`	⇔	`-3.45`	unaire notatie
`+(x,*(2,y))`	⇔	`x+2*y`	gangbare voorrangsregels
`*(x,+(2,y))`	⇔	`x*(2+y)`	

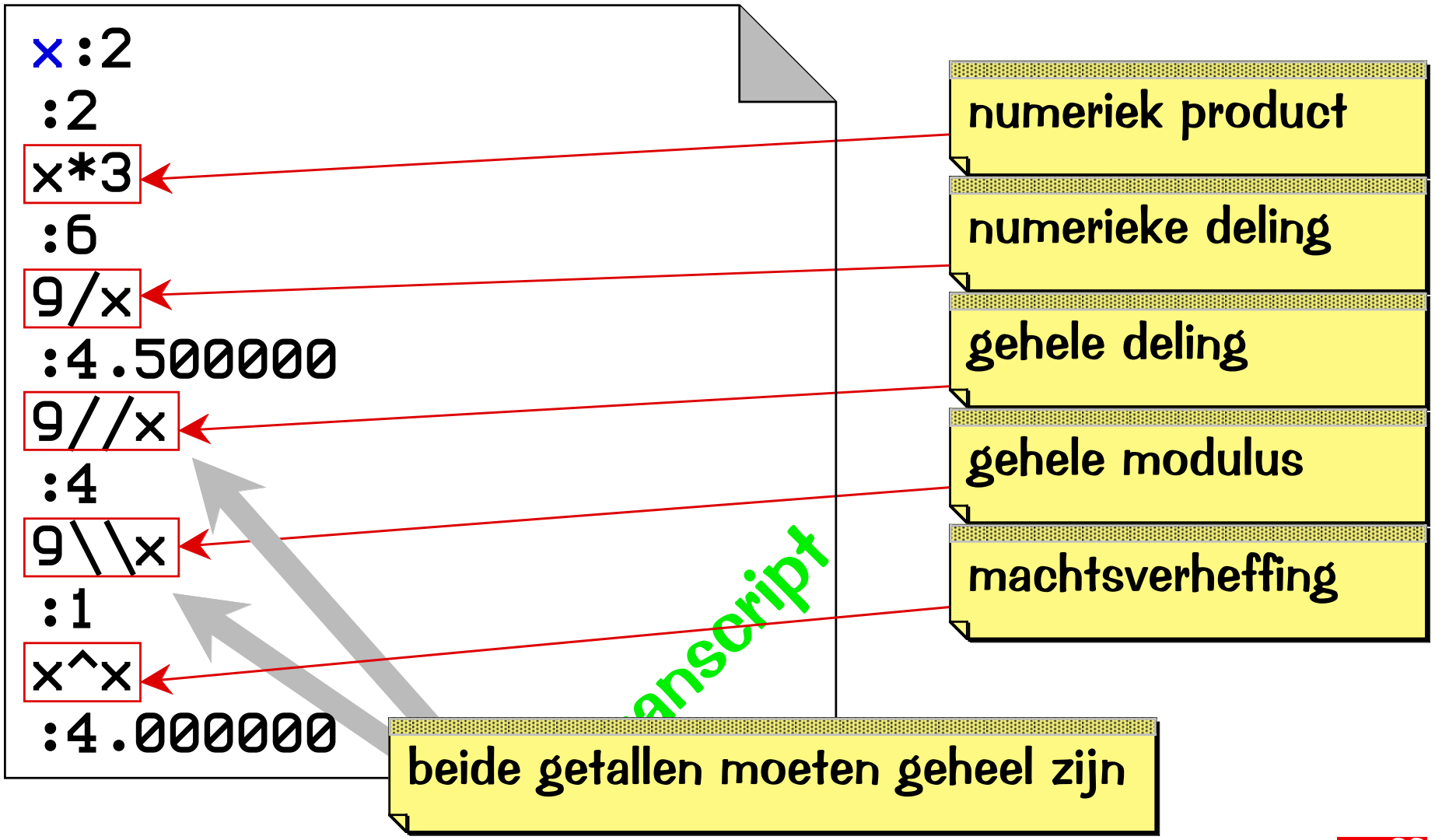
Bij aanvang is het woordenboek opgevuld met een reeks functies



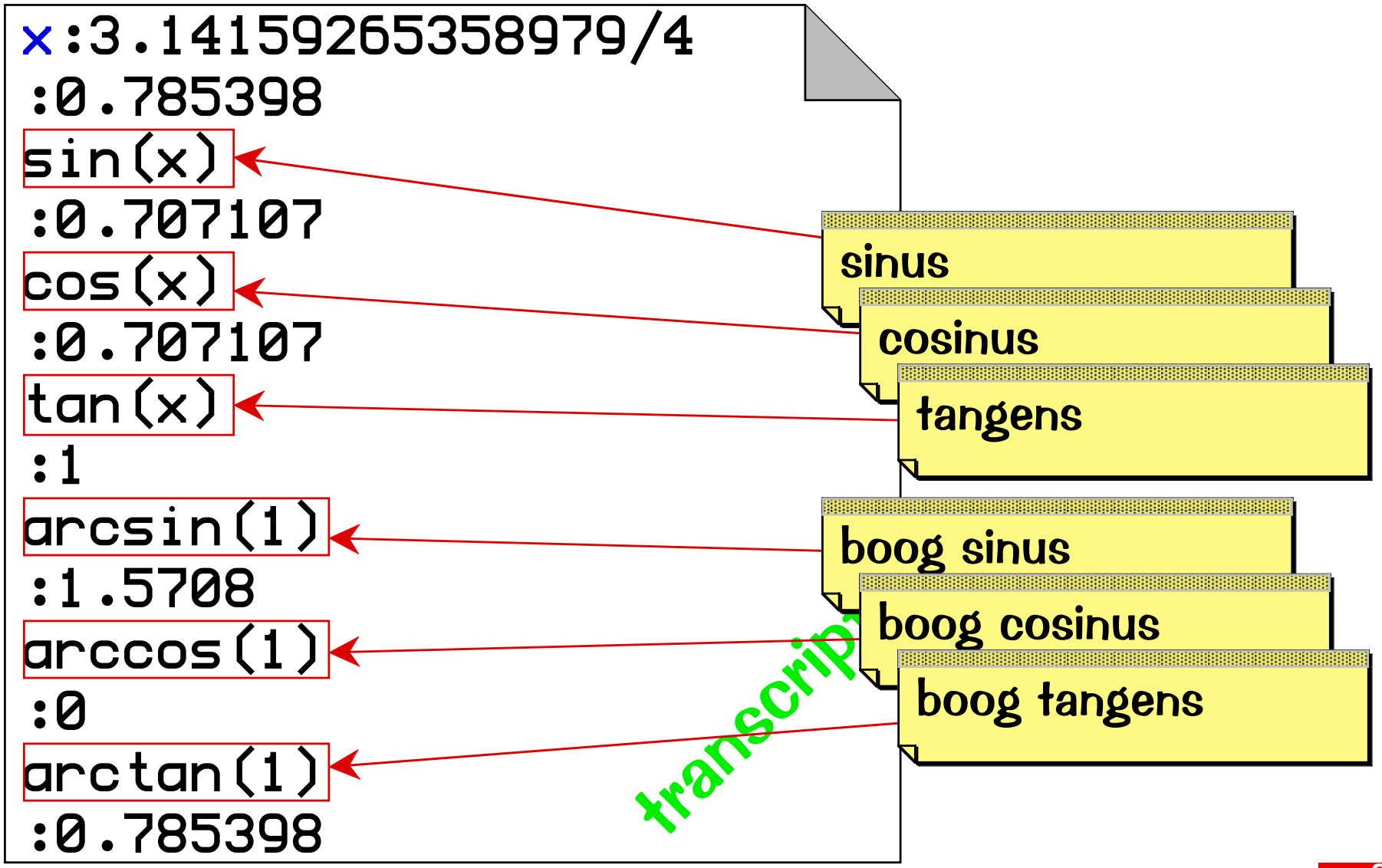
rekenkundige functies...



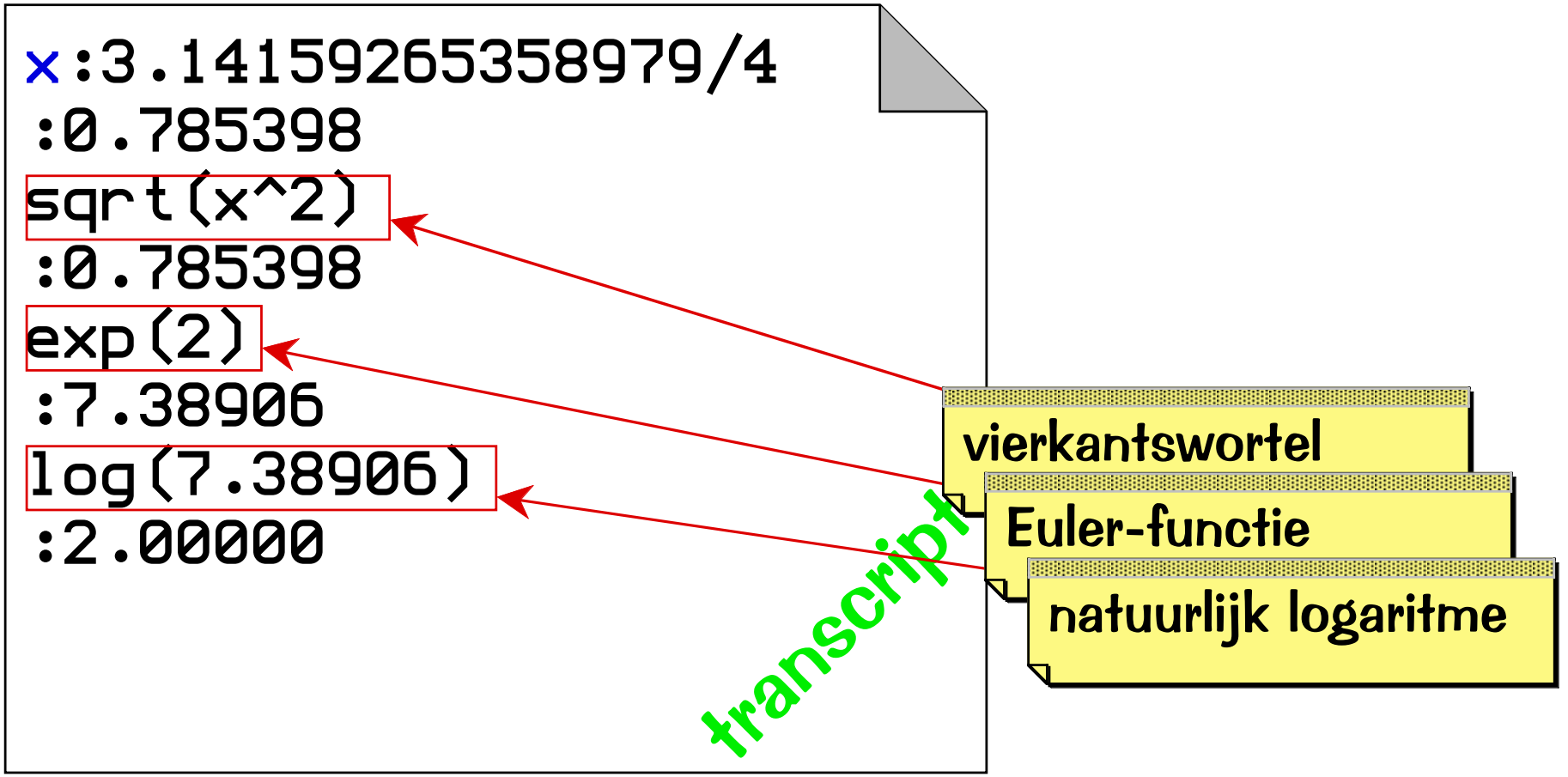
rekenkundige functies (vervolg)...



transcendente functies...



transcendente functies(vervolg)...



logische functies...

```
true
:<native function true>
false
:<native function false>
true(display('ok'),display('ko'))
:ok
false(display('ok'),display('ko'))
:ko
```

transcript

true(u_1,u_2) $\Rightarrow$ u_1

false(u_1,u_2) $\Rightarrow$ u_2

logische functies (vervolg)...

```
and(true,true)
:<native function true>
and(true,false)
:<native function false>
and(false,true)
:<native function false>
and(false,false)
:<native function false>
```

transcript

and(u_1,u_2) ⇒ u_1(u_2,false)

```
or(true,true)
:<native function true>
or(true,false)
:<native function true>
or(false,true)
:<native function true>
or(false,false)
:<native function false>
```

transcript

```
not(true)
:<native function false>
not(false)
:<native function true>
```

transcript

not(u) ⇒ u(false,true)

or(u_1,u_2) ⇒ u_1(true,u_2)

relationele functies...

```
123<456
:<native function true>
1.23>4.56
:<native function false>
1.0=1
:<native function true>
'abc'='def'
:<native function false>
'abc'>'def'
:<native function false>
'abc'<'def'
:<native function true>
equivalent(123,'abc')
:<native function false>
```

numerieke vergelijking

alfabetische vergelijking

generische vergelijking

uitvoer...

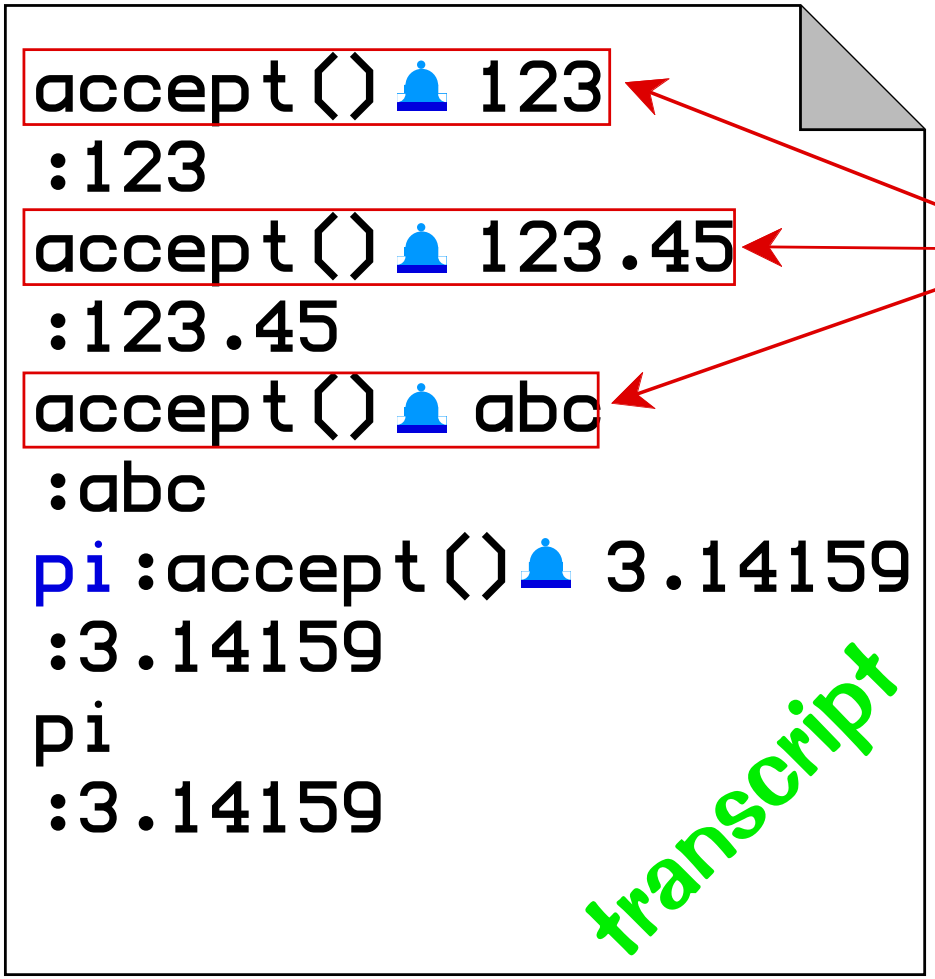
toon de waarden van de uitdrukkingen

```
display('abc',123)
 :abc123
display(3.1415926535897 9)
 :3.14159
display(true)
 :<native function true>
display(' x ',eoln,'xxx',eoln,' x ')
 : x
 :xxx
 : x
```

transcript

tekstuele voorstelling van de *einde-regel*

invoer...



maak tekst van de ingegeven tekens

= *beep*

tekst functies ...

```
t:'abcdef'
:abcdef
explode(t)
:[a, b, c, d, e, f]
implode(['x', 'y', 'z'])
:xyz
'hokus'+'pokus'
:hokuspokus
size('hokus'+'pokus')
:10
```

opsplitsen in lettertekens

samenvoegen van lettertekens

samenvoegen van tekst

lengte van een tekst

zie later!

transcript

controle functies: begin...

willekeurige uitdrukkingen

```
begin(display('naam?'),
      naam: accept(),
      eoln+'U heet '+naam)
:naam? Napoleon
:U heet Napoleon
{display('naam?');
 naam: accept();
 eoln+'U heet '+naam}
:naam? Bonaparte
:U heet Bonaparte
```

evalueer de uitdrukkingen van links naar rechts

geef de waarde van de laatste terug

transc

syntactisch equivalent:

begin(a,b,...,z) ≡ {a;b;...;z}

controle functies: if...

```
x: 123
:123
y:456
:456
if(x>y,x,y)
:456
if(x>y,x-y,y-x)
:333
if(x\\2=0,display('even'),display('oneven'))
:oneven
display(if(x\\2=0,'even','oneven'))
:oneven
```

if(*cond*,*then*,*else*)
⇒
cond(*then*,*else*)

transcript

controle functies: iteraties ...

```
k:1000
:1000
log2:0
:0
while((k:=k//2)>0,log2:=log2+1)
:9
```

transcript

```
{p:1;q:1;r:0}
:0
until(p>100000000,{r:=p+q;q:=p;p:=r})
:102334155
p/q
:1.61803
```

transcript

zie later!

```
x:1
:1
for(n:1,n:=n+1,not(n>10),x:=3*x)
:59049
3^10
:59049
```

transcript

tabel functies ...

aanmaak van een tabel

```
t:tab('a','e','i','o','u')
:[a, e, i, o, u]
z:accept()🔔 o
:o
for(i:1,i:=i+1,not(i>size(t)),
     if(t[i]=z,
        display(z,' is een klinker'),
        eoln))
:o is een klinker
:
t:=['a','e','i','o','u']
:[a, e, i, o, u]
```

zie later!

lengte van een tabel

syntactisch equivalent:

tab(a,b,...,z) ≡ [a,b,...,z]

transparant

type functies ...

```
is_number(1)
:<native function true>
is_fraction(2.4)
:<native function true>
is_text('abc')
:<native function true>
is_table([1,2,3])
:<native function true>
is_function(+)
:<native function true>
is_void(void)
:<native function true>
```

transcript

zie later!

conversie functies ...

```
trunc(1.2)
:1
char(97)
:a
ord('A')
:65
number('123')/2
:61.5
abs(-123)
:123
'n = '+text(123)
:n = 123
```

transcript

- geheel deel
- ordinaal->tekst
- tekst->ordinaal
- tekst->getal
- absolute waarde
- tekstuele waarde

zie later!

eigen functies ...

naam(*naam*$_1$, *naam*$_2$, ...) : *uitdrukking*

de *argumenten** van de functie

de *betekenis* van de functie

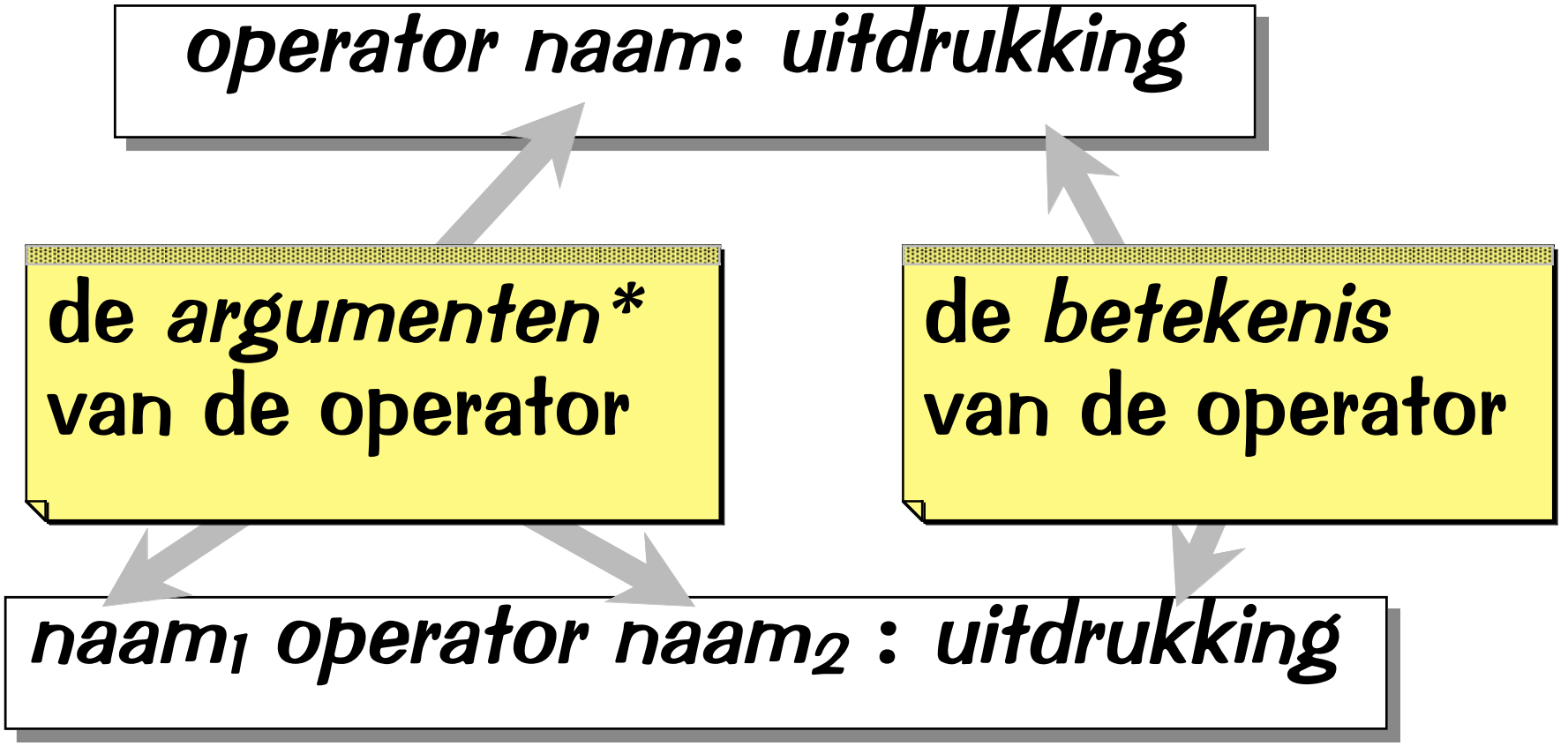
*voorlopig beperkt tot *namen*

eigen functies (vervolg) ...

```
gemiddelde(a,b): (a+b)/2
:<function gemiddelde>
gemiddelde(1,2)
:1.500000
maximum(x,y): if(x>y,x,y)
:<function maximum>
maximum(4,5)
:5
maximum(gemiddelde(1,5),gemiddelde(2,3))
:3.000000
```

transcript

eigen operatoren ...



*voorlopig beperkt tot *namen*

eigen operatoren (vervolg)...

```
\p:not(p)
:<function \>
p&q:and(p,q)
:<function &>
p|q:or(p,q)
:<function |>
{a:1;b:2;c:3}
:3
\(a=b)&(b<c)
:<native function true>
```

transcript

voorbeeld 1: de grootste gemene deler...

elementaire eigenschap: "als g de grootste gemene deler van p en q is, dan is g ook de grootste gemene deler van het minimum van p en q enerzijds, en het verschil tussen het maximum en het minimum van p en q anderzijds"*

$$ggd(p,q) = \begin{cases} ggd(p-q,q) & \text{als } p>q>0 \\ ggd(p,q-p) & \text{als } q>p>0 \\ p & \text{als } p=q>0 \end{cases}$$

* dit kan nog beter via de rest...

voorbeeld 1 (vervolg): de grootste gemene deler...

```
ggd(p,q):if(p>q,
            ggd(p-q,q),
            if(p<q,
               ggd(p,q-p),
               p))
:<function ggd>
ggd(121,33)
:11
```

transcript

voorbeeld 2: faculteit...

elementaire eigenschap: "als N de faculteit van n is, dan is N het produkt van n en de faculteit van n-1"

$$fac(n) = \begin{cases} n*fac(n-1) & \text{als } n>1 \\ 1 & \text{als } n\leq 1 \end{cases}$$

voorbeeld 2 (vervolg): faculteit...

```
fac(n):if(n>1,
          n*fac(n-1),
          1)
:<function fac>
fac(10)
:3628800
fac(12)
:479001600
```

transcript

groter dan 12
⇒ *overflow!*

voorbeeld 3: Fibonacci...

elementaire eigenschap: "een Fibonacci getal is steeds de som van zijn twee voorgangers"

$$fib(n) = \begin{cases} fib(n-1)+fib(n-2) & \text{als } n>1 \\ 1 & \text{als } n\leq 1 \end{cases}$$

voorbeeld 3 (vervolg): Fibonacci...

```
fib(n):if(n>1,
           fib(n-1)+fib(n-2),
           1)
:<function fib>
fib(5)
:8
fib(10)
:89
fib(20)
:10946
fib(30)
```

tijd $\rightarrow \infty$
want: fib(n)
neemt eens zoveel
tijd als fib(n-1)!

transcript

voorbeeld 3 (vervolg): Fibonacci...

```
fib(p,q,r):if(r>1,
                  fib(q,p+q,r-1),
                  q)
:<function fib>
fib(1,1,5)
:8
fib(1,1,10)
:89
fib(1,1,20)
:10946
fib(1,1,30)
:1346269
```

tijd is nu ok
want: fib(n) neemt
nu eens zoveel tijd
als fib(n//2)!

transcript

voorbeeld 4: vierkantsvergelijking...

```
L(a,b):
   {display('oplossing: ');
    if(a=0,if(b=0,'IR','geen'),-b/a)}
 :<function L>
L(0,0)
 :oplossing = IR
L(1,2)
 :oplossing = -2.000000
L(0,1)
 :oplossing = geen
```

eerste stap: coëfficiënt van x^2 is nul

transcript

voorbeeld 4 (vervolg): vierkantsvergelijking...

```
Q(a,b,c):
    if(a=0,
       L(b,c),
       {D: b^2-4*a*c;
        display('oplossing: ');
        if(D<0, display('geen'),
           if(D=0,-b/(2*a),
                  [(-b-sqrt(D))/(2*a),
                   (-b+sqrt(D))/(2*a)]))});
:<function Q>
```

tweede stap: coëfficiënt van x^2 is niet nul

transcript

voorbeeld 4 (vervolg): vierkantsvergelijking...

```
Q(1,0,0)
 :oplossing: 0.000000
Q(1,2,1)
 :oplossing: -1.000000
Q(1,-4,3)
 :oplossing:  [1.000000, 3.000000]
Q(0,0,0)
 :oplossing = IR
```

transcript