



Practicum Programmeerprincipes 2009-2010

fvdbergh@vub.ac.be

OPLOSSINGEN REEKS 2 SYSTEMATISCHE EXPLORATIE

Het waardenstelsel van Pico

Oefening 1. Voorspel telkens de uitvoer als je de functie toepast op de waarde.

Waarden:

void, 666, 666.666, "Hebban olla vogala nestas",
is_void, [13, "dertien"]

Functies:

is_void, is_fraction, is_number,
is_table, is_function, is_text

Geven true terug:

is_void(void), is_number(666), is_fraction(666.666), is_text("Hebban olla vogala nestas"), is_function(is_void) en is_table([13, "dertien"])

Het resultaat is telkens true of false en dit zijn ingebouwde functies.

```
> display([is_void(void),  
          is_number(666),  
          is_fraction(666.666),  
          is_text("Hebban olla vogala nestas"),  
          is_function(is_void),  
          is_table([13, "dertien"])])
```

```
[<function true>, <function true>, <function true>, <function true>, <function true>, <function true>]
```

```
> |
```

Gelijkheid van waarden

Oefening 2. Kies uit: texts, onberekenbaar, numbers, lexicografische, oneindig, fractions

Pico laat toe =, < en > toe te passen op numbers, fractions en op texts (merk op dat number = fraction lukt).

De orderrelatie op teksten is de gewone lexicografische orde.

Men kan = niet toepassen op functies omdat dit onberekenbaar is. Men zou namelijk een oneindig aantal waarden moeten testen.

Waardenomzeters

Oefening 3. Bepaal proefondervindelijk de betekenis en de types van de volgende unaire functies: trunc, char, ord, number, text, explode, implode.

Vul dan telkens in met $R = \{ r \mid r \text{ is een fraction} \}$, $N = \{ n \mid n \text{ is een number} \}$, $T = \{ t \mid t \text{ is een tabel} \}$, $X = \{ x \mid x \text{ is een tekst} \}$, $F = \{ f \mid f \text{ is een function} \}$, of een unie van deze verzamelingen.

trunc : $R \cup N \rightarrow N$
char : $\{n \text{ in } N \mid 0 \leq n \leq 255\} \rightarrow X$
ord : $\{x \text{ in } X \mid \text{size}(x)=1\} \rightarrow \{n \text{ in } N \mid 0 \leq n \leq 255\}$
number : $X \rightarrow N \cup \{\text{void}\} \cup R$
explode : $X \rightarrow T$
implode : $\{T \mid T[i] \text{ in } N\} \rightarrow X$

Ingebouwde operatoren

Oefening 4. Vind proefondervindelijk de argumenttypes en resultaattypes van de ingebouwde operatoren uit de volgende tabel. Merk op dat sommige operatoren zowel unair als binair voorkomen. Bepaal voor de twijfelgevallen ook de betekenis. In welke twee notationale stijlen kunnen deze operatoren gebruikt worden?

+ : $N \rightarrow N, R \rightarrow R$
- : $N \rightarrow N, R \rightarrow R$
+ : $N \times N \rightarrow N, N \times R \rightarrow R, R \times N \rightarrow R,$
 $R \times R \rightarrow R, X \times X \rightarrow X$
- : $N \times N \rightarrow N, N \times R \rightarrow R, R \times N \rightarrow R,$
 $R \times R \rightarrow R$
* : $N \times N \rightarrow N, N \times R \rightarrow R, R \times N \rightarrow R,$
 $R \times R \rightarrow R$
/ : $R \times R \rightarrow R, N \times R \rightarrow R, R \times N \rightarrow R$
// : $N \times N \rightarrow N$
\\ : $N \times N \rightarrow N$
^ : $R \times R \rightarrow R, N \times R \rightarrow R, R \times N \rightarrow R,$
 $N \times N \rightarrow R$

Ingebouwde functies

Oefening 5. Bepaal opnieuw de argument – en resultaattypes van de volgende ingebouwde transcendenten functies.

sin : $R \cup N \rightarrow R$ (of nauwkeuriger: $[1,-1]$)
cos : $R \cup N \rightarrow R$ (idem)
tan : $R \cup N \rightarrow R$
arcsin : $[-1,1] \text{ in } R \cup N \rightarrow R$ (of nauwkeuriger: $]-\pi/2,-\pi/2[$)
arccos : $[-1,1] \text{ in } R \cup N \rightarrow R$ (idem)
arctan : $R \cup N \rightarrow R$
sqrt : $R^+ \cup N^+ \rightarrow R^+$
exp : $R \cup N \rightarrow R$
log : $R^+ \cup N^+ \setminus \{0\} \rightarrow R$
table : $P(R \cup N \cup T \cup X \cup F \cup \{\text{void}\}) \rightarrow T$
begin : $P(R \cup N \cup T \cup X \cup F \cup \{\text{void}\}) \rightarrow R \cup N \cup T \cup X \cup F \cup \{\text{void}\}$
abs : $R \rightarrow R, N \rightarrow N$
size : $T \cup X \rightarrow N^+$

Oefening 6. De "eenheid" van de goniometrische functies is radiaal. Het logaritme is nepariaans. Het aantal argumenten van begin en tab is onbepaald.

Invoer en uitvoer

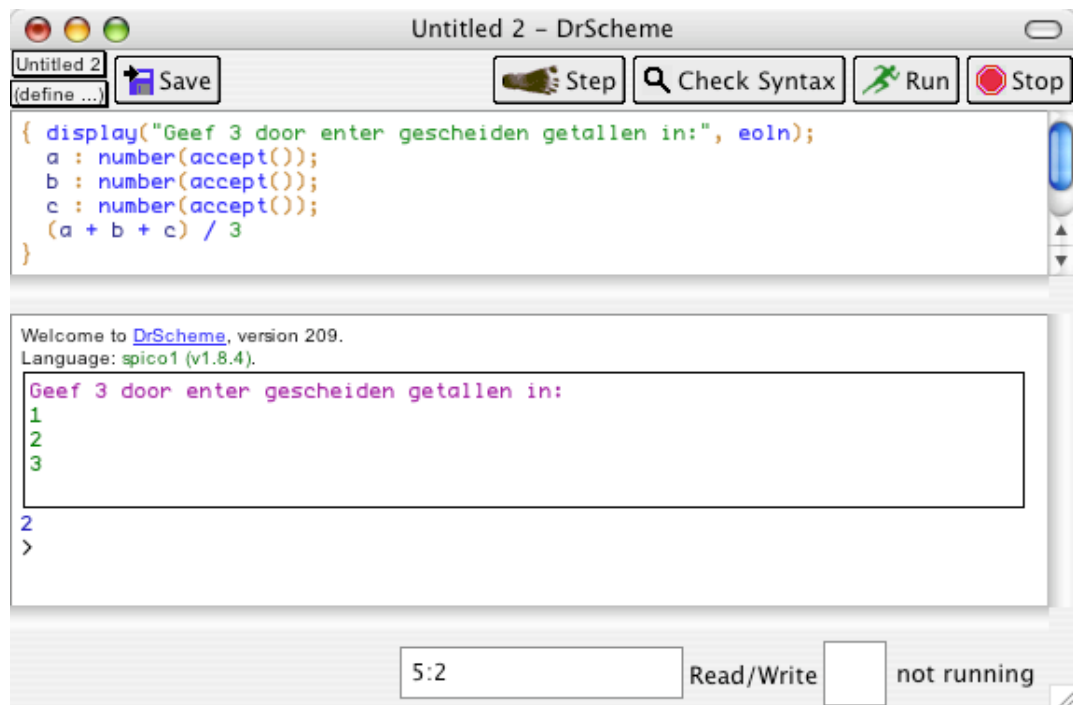
Oefening 7. Bepaal de types van volgende in – en uitvoer primitieven. Bepaal in detail wat deze functies doen.

`display` : $P(R U N U T U X U F U \{void\}) \rightarrow \{""\}$

`accept` : $\{ \} \rightarrow X$

`eoln` : dit is GEEN functie, maar een ingebouwde variabele met als waarde een TEKENREEKS die bestaat uit een lege lijn

Oefening 8. Neem het gemiddelde van 3 door de gebruiker ingegeven getallen. Gebruik hiervoor geen zelf gedefinieerde functies.



The screenshot shows the DrScheme IDE window titled "Untitled 2 - DrScheme". The code editor contains the following Scheme code:

```
{ display("Geef 3 door enter gescheiden getallen in:", eoln);  
  a : number(accept());  
  b : number(accept());  
  c : number(accept());  
  (a + b + c) / 3  
}
```

The output window shows the prompt "Geef 3 door enter gescheiden getallen in:" followed by the user input "1", "2", and "3" on separate lines. The prompt is followed by a new line and a greater-than sign (>).

At the bottom of the window, there is a status bar showing "5:2", "Read/Write", and "not running".

Oefening 9. Laat de gebruiker zijn naam, geboortedatum en moederland ingeven en print deze gegevens op een nette manier.

```
{  
  naam : { display("Naam?"); accept() };  
  datum : { display("Geboortedatum?"); accept() };  
  land : { display("Land?"); accept() };  
  display("Naam: " + naam + eoln  
    + "Land:" + land + eoln  
    + "Geboortedatum:" + datum)  
};
```

Syntactische suiker

Pico bevat nog wat extra syntactische suiker om het programmeren wat comfortabeler te maken. Zo werden accolades `{` en `}` toegevoegd (de expressies worden gescheiden door `;`) en werden vierkante haakjes toegevoegd om tabellen letterlijk op te schrijven (expressies gescheiden door `,`).

Oefening 10. Probeer te bedenken voor welke expressies de volgende Pico expressies slechts syntactische suiker bevatten. (Hint: ze staan voor de oproep van een functie).

```
{ e1; e2; e3; e4 } -> begin(e1, e2, e3, e4)
[ e1, e2, e3, e4 ] -> table(e1, e2, e3, e4)
```

Zelf functies maken

Volgende expressie definieert de identieke functie:

```
id(x): x
```

De functie herdefiniëren gebeurt door de toekenningsexpressie $id(x) := x * x$. Functies kunnen worden opgeroepen d.m.v. de gewone wiskundige notatie: $id(4)$.

Pico functies worden met een dubbele punt gedefinieerd. Verder is de syntaxis net dezelfde als in de wiskunde.

Je kan ook het functievoorschrift veranderen m.b.v. een toekenning .

Het oproepen van een functie gebeurt met de gewone wiskundige notatie.

Argumenten worden dus gescheiden door komma's .

Oefening 11. Implementeer functies die het volgende berekenen:

- het product van het verschil en de som van twee getallen
{
 $f(x,y) : (x-y) * (x+y);$
 $f(3,5)$
}
- de derde macht van een getal, zonder de $^$ operator te gebruiken
{
 $power3(x) : x * x * x;$
 $power3(3)$
}
- het gemiddelde van 3 getallen
{
 $mean(x,y,z) : (x + y + z)/3;$
 $mean(1,2,3)$
}

Keuzes maken

Oefening 12. Het zijn ingebouwde functies.

Oefening 13. True is een functie die de waarde van de evaluatie van het eerste argument teruggeeft, terwijl false de waarde van de evaluatie van het tweede argument teruggeeft.

Oefening 14. Deze definitie van true en false laat ons toe op een elegante manier de & (en), | (of) en ! (not) booleaanse functies te definiëren. De !(p) functie zal bijvoorbeeld zijn argument toepassen (we weten immers dat dit argument de functie true of false is) op de argumenten false en true:

```
> !p:p(false,true)
<function !>
> !true
<function false>
> !false
<function true>
> !(false)
<function true>
>
```

Kan je zelf definities voor & (en) en | (of) vinden? Noem deze mijn_en(p,q) en mijn_of(p,q).

```
{
  mijn_en(p,q) : p(q,false);
  mijn_of(p,q) : p(true, q)
}
```

Oefening 15. Er is duidelijk een verschil tussen het operationele gedrag van onze `my_and` en de ingebouwde `and` functie: `my_and` evalueert altijd zijn twee argumenten, terwijl `and` de evaluatie van het tweede argument overslaat als het eerste argument `false` is. Dit is een vorm van optimalisatie.

Oefening 16. Probeer nu een gelijkaardige transcript te maken waarmee je aantoonst dat er ook een operationeel verschil is tussen jouw `mijn_of` en de ingebouwde `|`: `mijn_of` evalueert altijd zijn twee argumenten, terwijl `|` de evaluatie van het tweede argument overslaat als het eerste argument `true` is. We zullen later nog zien hoe `|` dan wel in Pico geïmplementeerd is.

```
> {display("argument 1 evaluated", eoln); true} | {display("argument 2 evaluated", eoln); false}
argument 1 evaluated
<function true>
> mijn_of({display("argument 1 evaluated", eoln); true}, {display("argument 2 evaluated", eoln); false})
argument 1 evaluated
argument 2 evaluated
<function true>
```

Oefening 17. Kan je, analoog aan `mijn_en` en `mijn_of`, zelf een `my_if` definiëren?

Je zou het volgende kunnen proberen:
`mijn_if(conditie, dan, anders) : conditie(dan, anders)`

Dit is echter niet wat we willen want:
`mijn_if(false, display("niet"), display(" ok"))`

Later zien we nog hoe dit dan wel lukt.

Oefening 18. Definieer zelf een `less_than_or_equal`.

`x <= y : (x=y) | (x < y)`
of `x <= y : !(x > y)`

Oefening 19. Implementeer een functie van 3 argumenten die het grootste argument teruggeeft.

`biggest2(x,y) : if(y < x, x, y);`
`biggest3(x,y,z) : biggest2(biggest2(x,y), z)`

Oefening 20. Implementeer een exclusieve or.

`p $ q : (p | q) & !(p & q)`
of `p $ q : p(q(false,true),q)`

Oefening 21. Implementeer de logische implicatie en equivalentie.

`p => q : !p | q`

`p <=> q : (p & q) | (!p & !q)`
of `p <=> q : p(q,q(false, true))`

Merk op dat "=" niet gedefinieerd is op functies!

Oefening 22.

```
easter(year) : {
  a : year \\ 19;
  b : year \\ 4;
  c : year \\ 7;
  d : (19 * a + 24) \\ 30;
  e : (2 * b + 4 * c + 6 * d + 5) \\ 7;
  22 + d + e
}
```

Oefening 23.

```
extended_easter(year) : {
  day : easter(year);
  if((year = 1954) |
    (year = 1981) |
    (year=2049) |
    (year=2076),
    day - 7,
    day)
}
```

Jezelf herhalen

Oefening 24. Schrijf nu zelf een variant op bovenstaande herhaal(keer) functie die een aantal keer een door de gebruiker meegegeven tekst op het scherm zet.

```
herhaal(tekst, n) : if(n=0,
  display(""),
  {display(tekst);
  herhaal(tekst, n-1)})
```

Oefening 25. Implementeer een functie power(x,n) die zijn eerste argument tot de macht van het tweede argument verheft. Maak hiervoor geen gebruik van de ^ operator.

```
power(x,n) : if(n=0, 1, x * power(x, n-1))
```

Oefening 26. Schrijf ook een multiply-functie die geen gebruik maakt van de * operator.

```
multiply(x,n) : if(n=0, 0, x + multiply(x, n-1))
```

Oefening 27. Schrijf een functie die het n-de getal van Fibonacci teruggeeft.

```
{
  fib(n) : if(n<2, 1, fib(n-1) + fib(n-2));
  i : -1;
  numbers[10] : fib(i:=i+1)
}
```

(Bovenstaande fibonaccidefinities geeft het n-de getal van Fibonacci terug en begint te tellen vanaf 0; volgens deze definitie zijn het 0-de en 1-ste Fibonacci-getal 1)

Pico heeft ook 3 speciale expressies die het mogelijk maken om iets herhaaldelijks uit te voeren (een lus) zonder hiervoor een recursieve functie te moeten schrijven:

- while(conditie, expressie)
De while-lus zal de expressie uitvoeren zolang de conditie waar is.
- until(conditie, expressie)
De until-lus zal de expressie uitvoeren totdat de conditie waar is.
- for(initialisatie, conditie, stap, expressie)
Bij het begin van de for-lus zal de initialisatie-expressie uitgevoerd worden waarin typisch een waarde aan een iteratievariabele toegekend wordt. Als de de conditie-expressie waar is, zal de expressie uitgevoerd worden. Vervolgens wordt de stap-expressie uitgevoerd waarin typisch de waarde van de vooraf geïnitieerde iteratievariabele verhoogd of verlaagd wordt. Als de conditie nog steeds waar is, zal de expressie weer uitgevoerd worden, gevolgd door de stap-expressie enz ... Als de conditie niet langer waar is, stopt de lus.

Verklaar eveneens de waarden waarnaar volgende expressies evalueren:

```
> while(false, 3)
<void>
> until(true, 3)
3
```

Oefening 28. Bestudeer grondig het volgende transcript. Kan je de waarden van de namen z en teller doorheen het transcript verklaren?

Oefening 29. Implementeer het gedrag van de while-lus uit het voorbeeld met een for-lus. Doe hetzelfde voor de until-lus.

```
> teller:1
1
> for(teller:1, teller < 5, void, teller := teller + 1)
5
> for(teller, teller != 1, void, display(teller := teller-2, eoln))
3
1
>
```

Oefening 30. Implementeer nu op een iteratieve wijze (= door middel van lussen) de voorgaande oefeningen op recursiviteit.

```
herhaal(tekst, n) : {
  while(n>0, {display(tekst); n := n-1});
  ""
}

power(x,n) : {
  result:1;
  while(n!=0,{n := n - 1; result := result * x});
  result
}

power(x,n) : {
  result:1;
  for(i:n, i!=0, i := i-1, result := result * x);
  result
}

multiply(x,n) : {
  for(result : 0, n!=0, n:=n-1, result := result + x);
  result
}
```



```
}
```

Tabellen

Oefening 31. Evalueer de volgende expressie:

```
{ i:0; t[13]: i:=i+1; display(t) }
```

Toon het zesde element uit deze tabel en vervang het tiende element door het getal 13.

```
> {i:=0; t[13]:(i:=i+1); display(t)}
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
> t[6]
6
> display(t[6])
6
> t[10] := 13
<table>
> display(t)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 11, 12, 13]
> display(t[10])
13
> |
```

Er zijn 3 Pico uitdrukkingen waarmee je tabellen kan manipuleren.

- Met `t[idx]:exp` kan je een nieuwe tabel aanmaken van grootte `idx` en deze binden aan de naam `t` in het woordenboek. De `exp` uitdrukking wordt verschillende keren geëvalueerd, namelijk precies het aantal keren als wordt aangegeven door de waarde van `idx`.
- Het indexerend van tabel `t` met indexuitdrukking `exp` gebeurt door de `t[idx]` uitdrukking. De indices van een tabel `t` liggen tussen 1 en `size(t)`.
- Het veranderen van de waarde in tabel `t` op positie `idx` naar de waarde `exp` gebeurt met de uitdrukking `t[idx]:=exp`.

Oefening 32. Maak een tabel aan waarin alle elementen op een even plaats de waarde 1 en alle elementen op een oneven plaats de waarde 0 aannemen.

```
{
  i : 1;
  oddeven[10] : if(i=1, i:=0,i:=1)
}
```

of:

```
{
  i : -1;
  oddeven[10] : (i := i + 1) \ \ 2
}
```

Oefening 33. Maak een tabel aan waarin elk element overeenkomt met de faculteit van zijn positie.

```
{
  fac(x) : if(x=0, 1, x * fac(x-1));
  i:0;
  factab[10] : fac(i:=i+1)
}
```

```
{ i:0;
  fac:1;
  factab[10] : fac := fac * (i:=i+1)
}
```

Oefening 34. Maak de volgende 2-dimensionale tabel aan:

```
1 2 3 0
4 5 6 0
7 8 9 0
```

Ga na dat het derde getal van de tweede rij 6 is.

```
{
  matrix : [[1,2,3,0], [4,5,6,0], [7,8,9,0]];
  rij2 : matrix[2];
  rij2[3]
}
```

of

```
{
  i:0;
  matrix[3]:[i+1,i+2,i:=i+3,0];
  rij2:matrix[2];
  rij2[3]
}
```

of

```
{
  j1:0;
  j2:1;
  t2[3] : t3[4] : if(j2\4=0,{j2:=j2+1;0},{j2:=j2+1;j1:=j1+1})
}
```

Operatoren

De letters { <, =, >, #, ~ } U { +, -, |, \$, % } U { *, /, \, & } U { ^, !, ?, . } worden door Pico als speciaal behandeld. Functies waarvan de naam bestaat uit een combinatie van deze letters worden *operatoren* genoemd. De voorrangsregels voor de letters zijn:

{ ^, !, ?, . } > { *, /, \, & } > { +, -, |, \$, % } > { <, =, >, #, ~ }

Indien twee operatoren o1 en o2 uit meerdere letters bestaan, dan is
 $o1 > o2 \Leftrightarrow \text{eerste}(o1) > \text{eerste}(o2)$

Functies die aldus het statuut operator krijgen, kan je zowel in prefix als in infixnotatie definiëren en oproepen.

Oefening 35. Definieer nu de unaire operator !! zodat deze de faculteit van zijn argument berekent.

```
{
  !!x : if(x=0, 1, x * !!(x-1));
  !!5
}
```

Functies met een variabel aantal argumenten

Functies met een variabele aantal argumenten worden door de f@args constructie gedefinieerd. De argumenten waarmee f dan wordt opgeroepen worden als tabel aan de args variabele gebonden. De functie kan daarna met een willekeurig aantal argumenten opgeroepen worden.

Oefening 36. Implementeer een functie die het gemiddelde van zijn argumenten berekent.

```
{
  gemiddelde@getallen : {
    som : 0;
    for(i:1,!(i>size(getallen)),
      i:=i+1,som:=som+getallen[i]);
    som / size(getallen)
  };
  gemiddelde(1,2,3,4,5,6)
}
```

Oefening 37. Implementeer een functie die het grootste van zijn argumenten teruggeeft.

```
{
  grootste@getallen : {
    grootste : getallen[1];
    for(i:1, i:=i+1, i<size(getallen)+1,
      if(getallen[i]>grootste,
        grootste := getallen[i],
        void));
    grootste
  };
  grootste(1,2,3,8,4,2,0)
}
```

Oefening 38. Ga na dat de begin functie in Pico als volgt gedefinieerd kan worden:
begin@args:args[size(args)]

Oefening 39. Kan je zelf bedenken wat de definitie van de table functie is?

```
my_tab@args : args
```

