Practicum Programmeerprincipes 2006-2007

cderoove@vub.ac.be

REEKS 2
SYSTEMATISCHE EXPLORATIE

Deze oefeningenreeks is bedoeld om Pico wat beter te leren kennen op een systematische manier. In de volgende oefeningenreeksen zullen we op veel van de hier aangehaalde onderwerpen dieper ingaan.

Het waardenstelsel van Pico

Het woordenboek van Pico bestaat uit een verzameling van woord – waarde associaties. Dit woordenboek bevat na het opstarten van Pico reeds enkele woorddefinities; de ingebouwde functies.

Je kan dit woordenboek uiteraard ook uitbreiden met je eigen woorden en bijhorende waarden, maar daarvoor moet je eerst vetrouwd worden met de soorten van waarden die binnen Pico gebruikt mogen worden.

Pico heeft 6 soorten van waarden: void, fractions, number, texts, functions en tables. Binnen de informatica noemen we dit "types". Merk op dat numbers en fractions verschillende types zijn. Sommige ingebouwde functies (zoals de optelling) werken voor beiden, maar er zijn geen automatische conversies voorzien. Typematig bekeken zijn het dus verschillende dingen.

De bovenstaande waarden zijn primitief: alle andere waarden in Pico kunnen uit hen opgebouwd worden. De booleaanse waarden `true` en `false` zijn niet primitief. Ze zitten wel in Pico, maar werden als functies in Pico zelf geschreven. Hierover later meer.

Voor elk type is er een typecontrolefunctie die je kan toepassen op waarden en op variabelen die waarden bevatten. Je kan al deze functies toepassen op alle waarden.

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
```

Je moet dus 6 keer 6 experimenten doen. (Tip: Een functie `f` toepassen op de waarden `w1` en `w2` doe je door de gewone wiskundige notatie te gebruiken: `f(w1,w2)`).

Het resultaat is telkens .... of .... Wat is het type van deze twee waarden?

Gelijkheid van waarden

Laat ons even de binaire operatoren `=`, `<` , `>` inspecteren. Bepaal op welke waarden je de binaire operatoren mag toepassen en, indien het toegelaten is, wat dan wel de betekenis mag zijn van deze toepassing.

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

Pico laat toe `=`, `<` en `>` toe te passen op .... , ... en op ....
De orderelatie op teksten is de gewone .... orde.
Men kan = niet toepassen op functies omdat dit .... is. Men zou namelijk een .... aantal waarden moeten testen.

Waardenomzetters

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

Je kan onderstaande transcripts als leidraad gebruiken:

```
> trunc(4.3)
4
> trunc(4)
4
> trunc("vier")

***Traceback for error #30
***evaluating:
"vier"
***evaluating:
trunc("vier")

number or fraction argument required: trunc
>
```

```
> t : "1234"
1234
> is_text(t)
<function true>
> is_number(t)
<function false>
> n : number(t)
1234
> is_text(n)
<function false>
> is_number(n)
<function true>
> n := n +123
1357
> n := "xyz" + text(n)
xyz1357
> ord("a")
97
> char(97)
a
> test : trunc(-2.71)
-2
> abs(test)
2
>
```

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

```
trunc   : ...... →  ......
char    : {n in N | 0 =< n =< 255} →  X
ord     : ...... →  ......
number  : ...... →  ......
explode : ...... →  ......
implode : ...... →  ......
```

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 notationele stijlen kunnen deze operatoren gebruikt worden?

```
+       : ...... →  ......
-       : ...... →  ......
+       : N × N  →  N, N × R  →  R, R × N  →  R,
          R × R →  R, X × X  →  X
-       : ...... × ......  →  ......
*       : ...... × ......  →  ......
/       : ...... × ......  →  ......
//      : ...... × ......  →  ......
\\      : ...... × ......  →  ......
^       : ...... × ......  →  ......
```

Ingebouwde functies

Oefening 5. Bepaal opnieuw de argument – en resultaattypes van de volgende ingebouwe transcendente functies.

```
sin    : ...... →  ......
cos    : ...... →  ......
tan    : ...... →  ......
arcsin : ...... →  ......
arccos : ...... →  ......
arctan : ...... →  ......
sqrt   : ...... →  ......
exp    : ...... →  ......
log    : ...... →  ......
tab    : P(......) →  ......
begin  : P(......) →  ......
abs    : ...... →  ......
size   : ...... →  ......
```

Oefening 6. Wat is de eenheid van goniometrische functies? Welk logaritme is geïmplementeerd? Wat is het aantal argumenten van `begin`?

Invoer en uitvoer

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

```
> test:10
10
> tekst: accept()
hallo
hallo
> display("test = ", test, eoln, "tekst = ", tekst)
test = 10
tekst = hallo
> is_text(display(3))
3
<function true>
>
```

```
display    : ...... →  ......
accept     : ...... →  ......
eoln      : ......
```

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

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

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 }
[ e1, e2, e3, e4 ]
```

Vergewis je hiervan door de desbetreffende functies eens van een andere implementatie te voorzien.

Zelf functies maken

Volgende expressie definieert de identieke functie:

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

De functie herdefinieren 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 syntax net hetzelfde als in de wiskunde.
Je kan ook het functievoorschrift verandere m.b.v. een
....
Het oproepen van een functie gebeurt met de gewone wiskundige notatie.
Argumenten worden dus gescheiden door ....

Oefening 11. Implementeer functies die het volgende berekenen:

a. het product van het verschil en de som van twee getallen
b. de derde macht van een getal, zonder de ^ operator te gebruiken
c. het gemiddelde van 3 getallen

Keuzes maken

Oefening 12. We hebben reeds eerder de opmerking gemaakt dat de booleaanse waarden `true` en `false`, niet primitief zijn. Wat is dan wel type?

Oefening 13. Voer de volgende expressies uit: `true(1,2)`, `false(1,2)`, `true(false,true)`, `false(false, true)`.
True is een functie die zijn .... argument teruggeeft, terwijl false zijn .... 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

definieren. 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)`.

Oefening 15. Bestudeer aandachtig het volgende transcript.

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

Er is duidelijk een verschil tussen het operationele gedrag van onze `mijn_en` en de ingebouwde `&` functie: `mijn_en` evalueert ……………….. zijn twee argumenten, terwijl `&` de evaluatie van het tweede argument ……………….. als het eerste argument ……………….. is. Dit is een vorm van optimalisatie.

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

Oefening 17. Bij het programmeren moet er afhankelijk van de waarde van een conditie vaak een keuze gemaakt worden. Dit kan in Pico met de `if`-expressie die afhankelijk van de waarde van zijn eerste argument (de conditie), zijn tweede (de dan-expressie) dan wel zijn derde argument (de anders-expressie) evalueert:

if(conditie, dan, anders)

```
> a:5
5
> b:13
13
> if((a>b)|(a=b), a, b)
13
> if(is_text(a), a, text(a))
5
> |
```

Kan je, analoog aan `mijn_en` en `mijn_of`, zelf een `mijn_if` definiëren?

Oefening 18. Definieer zelf een functie `less_than_or_equal(x,y)` of de operator `x<=y`.

Oefening 19. Implementeer een functie `biggest2(x,y)` die het grootste argument teruggeeft. Implementeer daarna een functie `biggest3(x,y,z)` die het grootste argument teruggeeft gebruik makende van de functie `biggest2(x,y)`.

Oefening 20. Implementeer een exclusieve of-operator: `x$y`. De waarheidstabel van deze operator ziet er als volgt uit:

```
> display([true$true,true$false,false$true,false$false])
[<function false>, <function true>, <function true>, <function false>]
> |
```

Oefening 21. Implementeer de logische implicatie en equivalentie.

Oefening 22. Definieer een `easter(year)` functie die de dag van de maand berekent waarop Pasen valt.

Je kan tussen 1982 en 2048 de dag waarop Pasen valt als volgt berekenen:
a = rest van jaar gedeeld door 19
b = rest van jaar gedeeld door 4
c = rest van jaar gedeeld door 7
d = rest van (19 * a + 24) gedeeld door 30
e = rest van (2 * b + 4 * c + 6 * d + 5) gedeeld door 7
Pasen zou dan op (22 + d + e) Maart moeten vallen

Oefening 23. Je kan de vorige functie ook laten werken voor data tussen 1900 en 2099. In het jaar 1954, 1981, 2049 en 2076 zou Pasen 7 dagen voor de datum die de vorige functie teruggeeft moeten vallen.

Jezelf herhalen

Je hebt waarschijnlijk al wel een functie geschreven die zichzelf opnieuw oproept. Dit soort functies noemen we recursief: in de definitie gebruiken we het woord dat we proberen te definiëren. Het klassieke voorbeeld is de faculteitsfunctie:

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

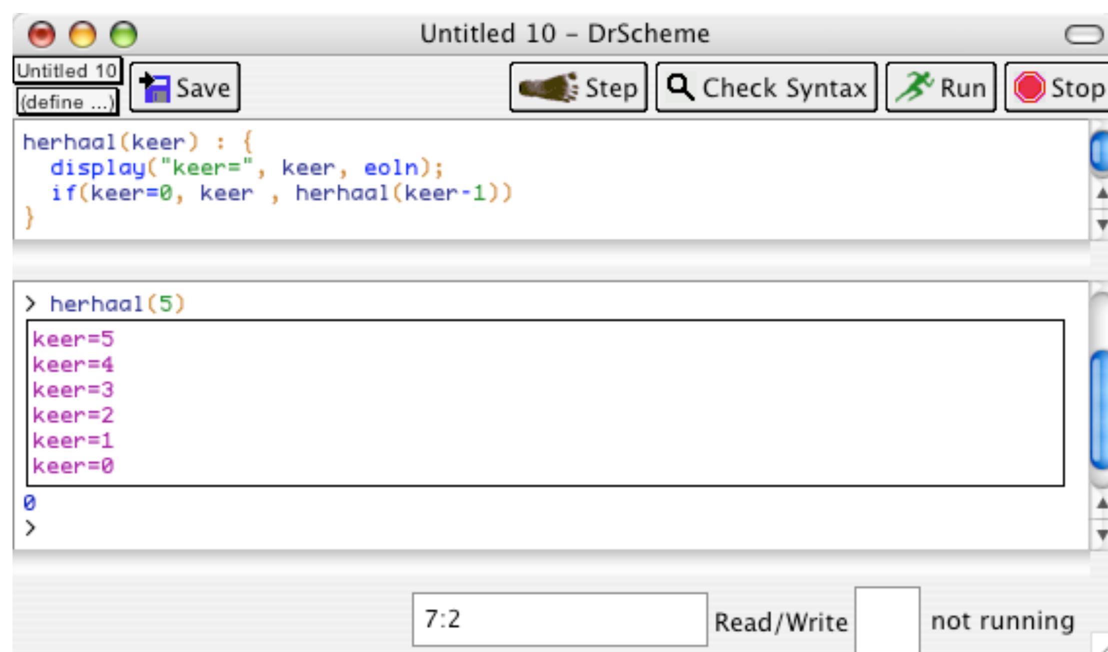
Merk hierbij op dat we steeds een stopconditie nodig hebben om ervoor te zorgen dat de functie uiteindelijk stopt en we dus een resultaat kunnen bekomen. Indien aan de stopconditie niet voldaan is, zal de functie opnieuw opgeroepen worden.

Het is dus ook mogelijk om functies te schrijven die nooit zullen stoppen:

```
blijflopen() : { display("*", eoln);
                 blijflopen() }
```

Je kan de evaluatie dan gelukkig nog stoppen in het menu Commands – Interrupt.

Dit soort constructies kunnen we uiteraard ook gebruiken om dingen herhaaldelijk uit te voeren:



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.

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

Oefening 26. Schrijf ook een `multiply(x,n)`-functie die geen gebruik maakt van de * operator.

Oefening 27. Schrijf een functie `fib(n)` die het n-de getal van Fibonacci teruggeeft.

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 .... de `conditie` waar is.
- `until(conditie, expressie)`
  De until-lus zal de `expressie` uitvoeren .... 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 geïnitialiseerde 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.

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

```
> teller:1
1
> while(teller < 5, teller := teller + 1)
5
> until(teller = 1, display(teller := teller-1, eoln))
4
3
2
1
> teller := teller + 1
2
> teller := teller + 5
7
> for(z:1,z<5,z:=z+1, display(teller \\ z, eoln))
0
1
1
3
> z
5
> teller
7
> |
```

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

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

Tabellen

Zoals je vroeger al gezien hebt, kan je tabellen aanmaken door de functie `table` toe te passen op de toekomstige elementen van de tabel. Je kan ook de vierkante haakjes als suikernotatie gebruiken:

```
> table(1,2,3)
<table>
> t : table(1,2,3)
<table>
> display(t)
[1, 2, 3]
> t := [4,5,6]
<table>
> display(t)
[4, 5, 6]
>
```

Een andere manier bestaat erin de tabel rechtstreeks te definieren met behulp van een `t[idx]:exp` waarbij een nieuwe tabel wordt gedefinieerd onder de naam `t`:

```
> t[5] : 0
<table>
> display(t)
[0, 0, 0, 0, 0]
> |
```

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.

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 ....
- Het indexeren van tabel `t` met indexuitdrukking `exp` gebeurt door de .... uitdrukking. De indices van een tabel `t` liggen tussen .... en ....
- Het veranderen van de waarde in tabel `t` op positie `idx` naar de waarde `exp` gebeurt door de uitdrukking ....

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.

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

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.

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 ⇔ eerste(o1) > 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.

Functies met een variabel aantal argumenten

Functies met een variabel aantal argumenten worden door de `f@args` constructie gedefinieerd. De argumenten waarmee `f` dan wordt opgeroepen worden als .... aan de `args` variabele gebonden (test dit!). De functie kan daarna met een willekeurig aantal argumenten opgeroepen worden.

```
> f@argumenten : display(argumenten)
<function f>
> f(1,2,3)
[1, 2, 3]
> f(2)
[2]
> f()
[]
>
```

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

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

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?