Third training school – RISC 2008

KANT/KASH tutorial

http://www.math.tu-berlin.de/~kant

LESSENI SYLLA

TU Berlin - Fakultät II

Institut für Mathematik Stra. des

17. Juni 136 D-10623 Berlin, Germany

lesseni(at)math.tu-berlin.de

Plan

Plan

- Introduction to KANT/KASH

Plan

- Introduction to KANT/KASH
- First steps in KASH3

Plan

- Introduction to KANT/KASH
- First steps in KASH3
- Applications

Plan

- Introduction to KANT/KASH
- First steps in KASH3
- Applications
- Programming Language

Programming Language

Programming Language

- function

Programming Language

- function
- if

Programming Language

- function
- if
- while

Programming Language

- function
- if
- while
- repeat

Programming Language

- function
- if
- while
- repeat
- for

Programming Language

- function
- if
- while
- repeat
- for
- Examples

Programming Language

KASH3 uses the GAP3 shell as a user interface. The programming language of GAP3 is an impreative language with some functional and some objects oriented features. In KASH3 additional features like Methods, Maps, and Extendable Objects are available.

function

Synthax:

$function([< arg - ident >, < arg - ident >])$
$[local < loc - ident >, < loc - ident >;]$
$< statements >$
$end;$

function

Purpose:

A function is in fact a statement; so it can be assigned to a variable or to a list element or a record component. Because for each of the formal arguments <arg-ident> and for each of the formal locals <loc-ident> a new variable is allocated when the function is called, it is possible that a function calls itself. This is usually called recursion. When a function <fun1> definition is evaluated inside another function <fun2>, KASH binds all the identifiers inside the function <fun1> that are identifiers of an argument or a local of <fun2> to the corresponding variable. This set of bindings is called the environment of the function <fun1>. When <fun1> is called, its body is executed in this environment.

function:

Example:

```
kash% addition:= function(arg1, arg2)
% #this function returns the sum of
% #the both arguments
% local a;
% a:= arg1+arg2;
% Print("The sum is:\n");
% return a;
% end;
```

function

NB:

A comfortable way to define a "simple" function is to used the maps-to operator: `->`

Examples:

```
cube:= x -> x^3;
cube(3)? cube(6.9)? cube(I)?
M:=Matrix(3,[2,8,9,5,4,0,1,3,2]);
cube(M)?

additionby5:= x -> x+5;
additionby5(0)? additionby5(-76)?
```

for

Synthax:

$for < variable > in < list > do < statements > od;$

Purpose:

The for loop executes the <statements> for every element of <list>. The statement sequence <variable> is first executed with <variable> bound to the first element of <list>, then with <variable> bound to the second element of <list> and so on. <variable> must be a simple variable, it must not be a list element selection or a record component selection.

for

Example:

```
kash% changelist:= function(L)
% #this function takes a list as
% #argument and changes its entries
% local i, K;
% K:= []; K[1]:= L[1];
% L[1]:= K[1]-2*L[Length(L)];
% for i in [2..Length(L)] do
%   K[i]:= L[i]; L[i]:= K[i]-2*K[i-1];
% od;
% return L;
% end;
```

What does it do?

if

Synthax:

$if < elt - alg\hat{}boo > then < statements1 >;$
$\{elif < elt - alg\hat{}boo > then < statements2 >\}$
$[else < statements3 >]$
$fi;$

if

Purpose:

The if statement allows one to execute statements depending on the value of some boolean expression. First the boolean expression following the if is evaluated. If it evaluates to true the statement sequence <statements1> after the first then is executed, and the execution of the if statement is complete. Otherwise the boolean expressions following elif are evaluated in turn. There may be any number of elif parts, possibly none at all. If the if expression and all, if any, elif expressions evaluate to false and there is an else part, which is optional, its statement sequence <statements3> is executed and the execution of the if statement is complete. The if statement terminates by a fi keyword.

if

Example:

```
kash% checkprimenumber:= function(a)
% #this function checks if the given
% #number is prime or not!
% if IsPrime(a) then
%    Print(a);
%    Print(" is a prime number \n");
% else
%    Print("Not, bye big loser!\n");
% fi;
% end;
```

while

synthax:

$while < elt - alg\hat{}boo > do < statements > od;$

Purpose:

The while loop executes the <statements> while the condition evaluates to true. First the boolean expression is evaluated. If it evaluates to false execution of the while loop terminates and the statement immediately following the while loop is executed next. Otherwise if it evaluates to true the <statements> are executed and the whole process begins again.

repeat

Synthax:

$repeat < statements > until < elt - alg\hat{}boo >;$

Purpose:

The repeat loop executes the statement sequence <statements> until the condition evaluates to true. First <statements> are executed. Then the boolean expression is evaluated. If it evaluates to true the repeat loop terminates and the statement immediately following the repeat loop is executed next. Otherwise if it evaluates to false the whole process begins again with the execution of the <statements>.

Examples

Example 1

```
kash% Mul_and_Inv:= function(arg1)
% #this function returns a map that
% #multiplies by "arg1"
% local phi, psi;
% phi:= function(arg2)
%   return arg2*arg1; end;
% psi:= function(arg3)
%   return arg3/arg1; end;
% return Map(Q, Q, phi, psi);
% end;
```

Examples

Example 2

```
kash% gcd_lcm:= function(arg1, arg2)
% #this function returns the GCD
% #and the LCM of the both arguments
% local a, b;
% a:= GCD(arg1, arg2);
% b:= LCM(arg1, arg2);
% return [a, b];
% end;
```

Examples

Example 3

```
kash% gcd_int:= function(arg)
% #this function returns the GCD
% #of the given integers.
% local c, r, how, i;
% if Length(arg)=1 then return arg[1];
% elif Length(arg) <> 1 then
%   how:= function(a, b)
%    while b <> 0 do
%    r:= b; b:= a mod b; a:= r; od;
%    return a;
%   end;
%   c:= how(arg[1], arg[2]);
%   for i in [3..Length(arg)] do
```

Examples

```
%   c:= how(c, arg[i]); od;
%   return c;
% fi;
% end;
```

Examples

Example 4

```
kash% GCD_int:= function(arg)
% #this function returns the GCD
% #of the given integers.
% local c, r, how, i;
% if Length(arg)=1 then return arg[1];
% elif Length(arg) <> 1 then
%   how:= function(a, b)
%    repeat r:= a mod b; a:= b; b:= r;
%    until b=0;
%    return a;
%   end;
%   c:= how(arg[1], arg[2]);
%   for i in [3..Length(arg)] do
```

Examples

```
%   c:= how(c, arg[i]); od;
%   return c;
% fi;
% end;
```

Examples

Example 5: The program file prog.k

```
InverseMatrix:= function(M)
#this function returns the inverse
#of a matrix over Z if it is
#invertible using the formula:
#Inv(M) = Adjoint(M)*(1/Det(M))
local A;
A:= Determinant(M);
if BaseRing(M) <> Z then
return "The coeff are not in Z,bye!";
elif A = -1 or A = 1 then
 Print("The inverse is:  \n");
 Print(Adjoint(M)*(1/A), "\n");
else
```

Examples

```
  Print("The determinant is:  ");
  Print(Determinant(M), "\n");
  Print("Not invertible over Z! \n");
 fi;
 end;
```

How to load the program file prog.k in KASH?

```
kash%  Read("prog.k");
```

Examples

Exercises

1) Create a function named `mult_inv` with domain $\mathbb{R}$ and codomain $\mathbb{C}$ which returns a map that multiplies by `"I*arg1"` and also compute the preimages (ref. example 1).

2) Create a function named `primes` which returns the list of prime numbers less or equal to the given argument.

Thank you for your attention!