K Variable Types

k_variable_types.k

```
/ k variable types
/  schreck20120221

/ Variables Types
/ 1. Local
/ 2. Global
/ 3. Parent-Local

/ SECTION 1. Local Variables
/    are only available to the function they are local to.
/    A variable is local to a function if:
/     A. It is an argument to the function.
/     B. There is a complete assignment inside
/        the function.
/     C. It is the root element of a dot assignment
/        inside the function.
/    notes:
/     *  Complete assigment means strickly token:value
/        no brackets or dots allowed. The token to the
/        left of the bracket must be something that
/        can be used as a function argument.
/          valid: f:{[a]}
/          invalid: f:{[a.b]}; f:{[a[3]]}
/     *  Dot assignment mean the left argument to the
/        assignment colon is a dot expression but not
/        and absolute reference, or it does not start
/        with a dot.
/     *  There are a few more complicated assigments
/        that do not cause localization. If there are
/        brackets or operators to the left of the : or
/        you are using ammend functions (3/4-adic . and @)
/        then localization will not happend
/     *  Functions defined within other functions have
/        limited access to their parent's locals. Covered
/        in section 3.

/
/ ex 1-1:complete assigment, rule 1A
{
    a:0 / localizes a
    :a / is available to this function
    }[]
/ 0
`a _in !.k / was not globalized
/ 0

/
/ ex 1-2:argument, rule 1B
{[a] / localizes a
    :a / is available to this function
    }[1]
/ 1
`a _in !.k / not globalized
/ 0


/
/ ex 1-3:localization set values prior to assignment
/        to _n
{
    before:b
    b:3
    after:b
    (before;after)}[]
/ (;3)
`b _in !.k / not globalized
/ 0
/ ex 1-4:localization with dot assignment
{
    u.a:1 / localized u as u:.,(`a;1;)
    u}[]
/ ,.(`a;1;)
`u _in !.k / not globalized
/ 0

/ ex 1-5:calls from before localization by
/        dot assingment are null
{
    before:u
    u.a:1 / localized u
    after:u
    (before;after)}[]
/ (;.,(`a;1;))

/ ex 1-6:Localization by assignment really acts like
/        reassigning an argument that was passed initially
/        as null
{[  args]
    before:a
    a:1;
    after:a
    (before;after)}[]
/ (;1)
{[a;args]
    before:a
    a:1
    after:a
    (before;after)}[_n][]
/ (;1)

/ SECTION 2. Global Variables
/    are available to any function via the k tree.
/    Globals can be referenced either relatively or
/    absolutely. A variable is global if it is:
/     A. Relatively assigned outside of a function.
/     B. Absolutely assigned anywhere.
/     C. Referenced in a function where it is not local.
/     D. Relatively assigned with (::), unless its
/        been localized anywhere else in the same
/        function.
/    notes:
/     *  Reference in a function, anything that isn't
/        locally available and you are making it global
/        at _d defaulting to _n value. _d is where
/        you were when you assigned or anonymously
/        executed the function. rule 2C.
/     *  Any kind of non-localizing assignment will
/        affect the global.

/ ex 2-1:assigning absolute references in a function
/        will always change the global rule 2B
a:2 3 4 / globalizes a, rule 1A
`a _in !.k
/ 1
{
    .k.a:,("something new";10) / globally sets .k.a, rule 1B
    }[]
a~,("something new";10)
/ 1

/ ex 2-2:retrieving global values in functions.
/ relative, rule 2A
a:2
f:{a} / points to the global .k.a
f[]
/ 2
a:3
f[] / still points to .k.a even after it changed
```

```
/ 3

/ ex 2-3:absolute, rule 2B
.k.a:2
f:{.k.a}
f[]
/ 2
.k.a:3
f[]
/ 3

/ ex 2-4:completely reassigning relative globals in
/        a function is only possible with ::, rule 2D
a:"string"
{a::3}[]
a
/ 3

/ ex 2-5:assigning indices of relative global
/        vectors/dicts in a function is possible
/        with vector[indices]:new_vals notation
a:2 3
{a[0]:-2}[]
a
/ -2 3
{a[1]:4 5}[]
a
/ (-2;4 5)
a:.()
a.b:1
a.c:2
{a.b}[]
/ 1
a
/ .((`a;1;);(`c;2;))

/ ex 2-6:localization overrides globals, rule 1A,2A
a:2
f:{
    a:(99 98;"string";,,,,`vvvvvvv) / localizes a
    a}
f[]
/ (99 98;"string";,,,,`vvvvvvv)
a
/ 2

/ ex 2-7:any complete relative reassignment will
/        localize no matter the order, the variable
/        we be _n prior
a:3 4
f:{
    a[0]:-3 / type error, a is _n
    a:2
    a}
.[f;_n;:]
/ (1;"type")
a
/ 3 4

/ SECTION 3. Parent-Local Variables
/    Functions defined within other functions
/    have the normal expected behavior with
/    access and modification of globals and
/    locals as explained in the first two
/    sections. However, they also have limited
/    access to the locals that were created
/    in its parent function. These will not act
/    the same as a normally localized variable.
/      A. Locals passed to a child are unavailable
/         to grand-children.
/      B. They are not dot-notation accessible by
/         the child.
/    These two subtleties can cause big problems,
/    avoid parent-local variables.

/ ex 3-1:normal local-like behavior
a:`global
{[a] / parent has localized a
    out.parent:a
    out.child:{a}[] / child function has a parent-local a~`local
    out}[`local]
/.((`parent;`local;)
/  (`child;`local;))

/ ex 3-2:rule 3A
a:`global
{[a] / local to parent
    out.parent:a
    out.child:{a}[]
    out.grandchild:{{a}[]}[]
    out}[`local]
/.((`parent;`local;)
/  (`child;`local;)
/  (`grandchild;`global;))

/ ex 3-3:rule 3B
a:.()
a.b:`global
{[a]
    out.parent:a.b
    out.child:{a.b}[]
    out}[.,(`b;`local;)]
/.((`parent;`local;)
/  (`child;`global;))

/ ex 3-4:more local-like behavior
{[a]
    a[0]:1 / a is local, but updated to a~1 0
    child:{
        / child has a parent-local a~1 0
        a[1]:2 / and updates it to a~1 2
        a}[]
    / parent local remains a~1 0
    (child;a)}[0 0]
/ (1 2;1 0)

/ ex 3-5:more complex 3B
{[a]
    a.b:1 / a is local, but updated to a~.((`b;1;);(`c;0;))
    out.child:{
        / child could have a parent-local a~.((`b;1;);(`c;0;))
        a.c:2 / but does not since this localizes a
        a}[]
    / parent local remains a~1 0
    out.parent:a
    out}[.((`b;0);(`c;0))]
/.((`child
/   .,(`c;2;)
/   )
/  (`parent
/   .((`b;1;)
/     (`c;0;))
/   ))

/ ex 3-6:3-5 with brackets
{[a]
    a[`b]:1 / a is local, but updated to a~.((`b;1;);(`c;0;))
    out.child:{
        / child could have a parent-local a~.((`b;1;);(`c;0;))
        a[`c]:2 / this does not localize a like ex 3-4
        a}[]
    / parent local remains a~1 0
    out.parent:a
    out}[.((`b;0);(`c;0))]
```

```
/.((`child
/   .((`b;1;)
/     (`c;2;))
/   )
/  (`parent
/   .((`b;1;)
/     (`c;0;))
/   ))

/ ex 3-7:3B can cause errors
a:1 / global a is an integer, cannot be dot-notation
{
    a.b:1 / local a~.,(`b;1)
    {{a.b}[]}[]
    }
/ this function cannot even be defined, parse error
```