mcode core

```
// mcode core for javascript
// see "Notes About the Core file" in the primer

let‗ r,ver = 'mcode core version 0.08.01.2024'
'' ⎕ ver ;  ⊙.test = {} // test artifacts namespace

r = `// mcode core - generated file
mcode.logn('` + ver + ` loaded from cache');
/* jshint asi:true */
`

r += '// Construction functions'
r += ⍎.er `
∇ mcode.create : r                     // ⍙    create non-literal and non-built-in types, add primitives
    // ⎕ 'create' ; ⎕.j α ; ⎕.j ω ; ⎕.j δ

    // vars:     ⍙.v varlist          outputs  let varlist
    // classes: [ args ] ⍙.n Class  outputs  new Class(...[args])

    ∇ mtx : i=0,j,q,t
        r = []
        i < α[0] ⌸
            j = 0 ; q = []
            j < α[1] ⌸
                t = δ=='i'?(i==j?1:0):0     // .i for identity matrix else zero matrix
                q.push(t) ; j++
            r.push(q)
            i++
        ⍔ r
    ∇ vec : i=0
        r = []
        i < α ⌸
            r = r.concat(0)
            i++
        ⍔ r

    r = ω
    ω ↯ '@'          → ⍔ (α!=null)?new‗ Date(α):new‗ Date()  // nb. ternary operator used ? :
    ω ↯ '{}'         → ⍔ (α!=θ)?new‗ Map(α):new‗ Map()       // to init Map, α is [[k v]...]
    ω ↯ '/'          → ⍔ new‗ RegExp(α,δ)                    // regex, δ are flags
    ω ↯ '[[]]'       →                                       // create matrix of shape [α]  .i for identity
        ( α instanceof‗ Array ) → ⍔ mtx(α,ω,δ)
        ⍔ [[]]                                               // empty matrix
    ω ⇋ '[]' →
        α != ⍬      → ⍔ vec(α,ω,δ)
        ⍔ []
    ω ⇋   '#'       → ⍔ parseInt(α)                        // convert to integer

    δ ⇋   'j'       → ⍔ JSON.parse(ω)                       // .j for JSON to data
        // todo: JSON.parse with replacer & try/catch
    α != ⍬          → ⍔ mcode.addPrim(α,ω,δ)                // if α given, then define primitive

    // ⋄ ( ( typeof‗ window[ω] ) ⇋ 'function' ) ^ α != ⍬ →
    // ⋄ ( ω instanceof‗ Function ) ^ α != ⍬ →
        // mcode.addPrim(α,ω,δ)
    // todo: table      an object with a field list α, a format string (TBD), data fields
    ⍔ 0
mcode.addPrim('⍙','mcode.create')   // R arg is quoted in transpiler fn 'OPsub'
`


r += ⌽̲.er `
∇ mcode.format0                     // ⌽̅    convert to string or JSON
    δ=='j' → ⍔ JSON.stringify(ω)
    ⍔ ω+''
`
/*
not added as primitive since OPsub routes ⌽̅.mod calls to format0
non-modified format ⌽̅ is handled by vf, see fmapIn
todo:
string ops: toString, pad, trim, justify     // no α but δ given
.s sprintf style  ω is VM (α for each element), table (α for each row)
*/

⌽̲ `
∇ ⊙.test.create
    // ⎕ '⊙.test.create'
    123 != '123px' ⍙ #                          → throw‗ 'error: ⍙ # test failed'
    ⍙.v m,r
    r = '"1970-01-01T00:00:00.000Z"'
    r != ⌽̅.j 0 ⍙ @                              → throw‗ 'error: ⍙ @ test failed'
    m = [['a',0],['b',1]] ⍙ {}
    1 != m.get('b')                             → throw‗ 'error: ⍙ {} test failed'
    r = 'abc' ⍙.g '/'
    '/abc/g' != r.toString()                    → throw‗ "error: ⍙ '\\/' failed"
    '[[]]' != ⌽̅.j ⍙ [[]]                        → throw‗ 'error: ⍙ [[]] failed'
    '[[0,0,0],[0,0,0]]' != ⌽̅.j [ 2 3 ] ⍙ [[]]   → throw‗ 'error: [] ⍙ [[]] failed'
    m = [ 3 3 ] ⍙.i [[]]
    '[[1,0,0],[0,1,0],[0,0,1]]' != ⌽̅.j m        → throw‗ 'error: [] ⍙.i [[]] failed'
⊙.test.create 0
```

```
`

r += ϕ.er `
▽ mcode.typeof : r=1,s      // ⍷    ⍷ ω   or   α ⍷ literal  true if α is typeof ω  [?] V or M

    // ⎕ 'typeof' ; ⎕ α ; ⎕ ω ; 'ω.length' ⎕ ω.length // debug

    α!=null                         → {s=ω;ω=α;α=s} // swap for comparisons below

    // ( α=='[?]' ) ^ ( typeof_ ω.length ) == 'number' → ⍔ 1   // true if α is indexed [] string or Array
    ( α=='[?]' ) ^ ( ω instanceof_ Array ) → ⍔ 1    // true if α is any array type
    ω ⇋ undefined                   → r = 'U'       // undefined   usually an error
    ⋄ ω ⇋ null                      → r = 'θ'       // null        means 'nothing'
    ⋄ ( ω instanceof_ Array )       →
        ω[0] instanceof_ Array      → r = '[[]]'    // matrix
        ⋄ r = '[]'                                  // vector
    ⋄ ω instanceof_ Map             → r = '{}'      // map (aka dictionary)
    // todo: table  r = '⍑'
    // below are scalar types:
    ⋄ ω instanceof_ Function        → r = '()'      // function
    ⋄ ω instanceof_ Date            → r = '@'       // date
    ⋄ ( typeof_ ω ) ⇋ 'boolean'     → r = '~'       // boolean
    ⋄ ( typeof_ ω ) ⇋ 'number'      → r = '#'       // number
    ⋄ ω instanceof_ RegExp          → r = '/'       // regex
    ⋄ ( typeof_ ω ) ⇋ 'string'      → r = ''        // string  nb. does not show unless as JSON or quoted
    ⋄ ( typeof_ ω ) ⇋ 'object'      → r = '.'       // object
    // ⋄ isObj ω
    α!=θ → r = 0 + ( α ⇋ r )                        // 1 if typeof α is symbol ω
    // ⎕ r // debug
    ⍔ r
    // ▽ isObj : p=θ    // NIU getPrototypeof FAILS on DOM objects
    //     ( typeof_ω ) ⇋ 'object' → p=Object.getPrototypeof(ω)
    //     ⍔ ω ^ ( ( ( p ⇋ null ) ∨ p ⇋ Object.prototype )
    // ▽ isObj : p=Object.getPrototypeof(ω)
    //     ⍔ ω ^ ( ( typeof_ω ) ⇋ 'object' ) ^ ( ( p ⇋ null ) ∨ p ⇋ Object.prototype )
'⍷' ⍙ mcode.typeof              // R arg is quoted for null ⍷ L in transpiler fn 'OPsub'
`

r += ϕ.er `
▽ mcode.system : p,rs                               // system functions  set by modifier δ
    // many of these operations are specific to JavaScript in the Browser environment and the mcode IDE
    δ == θ          → ⎕ mcode.guide()
    ⋄ δ=='d'        → debugger                      // ⍠.d 0  calls debugger
    ⋄ δ=='↑'        → throw_ 'error: ' + ω          // ⍠.↑  we're outta here via exception
    ⋄ δ=='o'        → mcode.shellOpts = ω           // see 'mcodeOptions.debug' in mcode.js
    ⋄ δ=='a'        → mcode.assertOpts = ω          // see ?=  mcode.assert
    ⋄ δ=='m'         → mcode.msg(ω)                 // 'error' or 'ready' for mcode_ide.js
    ⋄ δ=='tmo'       → setTimeout(ω)                // call function ω after all other processing is done
    ⋄ δ=='timer'     →                              // make async caller function wait
      ⎕ 'timer ' + ω + 'ms'
      ⍔ new‗Promise‗(rs=>setTimeout(()=>rs(0),ω)) // use await‗ in ∇.a (async) fn
    ⋄ δ=='busy'      → mcode.setBusy()              // creates promise for IDE
    ⋄ δ=='done'      → mcode.done(ω)                // resolves promise, ω is return data
    ⋄ δ=='wait'      →                              // make IDE wait
      mcode.setBusy()
      setTimeout(()=>mcode.done('waited '+ω+'ms'),ω)
'⍠' ⍙ mcode.system
`


⏀ `
∇ ⊙.test.typeof : V,M,x=0,y='abc',p={}
    // ⎕ '⊙.test.typeof'
    ~ x ⍷ #             → ⍠.↑ '⍷ test failed on #'
    ~ y ⍷ ''            → ⍠.↑ '⍷ test failed on '' var'
    ~ p ⍷ .             → ⍠.↑ '⍷ test failed on object'

    V = [ 0 1 2 ]
    '[]' != ⍷ V         → ⍠.↑ '⍷ test failed on []'

    M ← [   'a', 0
            'b', 1 ]
    '[[]]' != ⍷ M       → ⍠.↑ '⍷ test failed on [[]]'

    1 != M ⍷ [?]        → ⍠.↑ '⍷ test failed on [?]'
⊙.test.typeof 0
`


r += ⏀.er `
∇ mcode.shape0                              // ρ    simple length ω  if ω is []
    ω ⍷ [] → ⍔ ω.length                     //      nb. full ρ is defined later
    ω ⍷ [[]] → ⍔ [ω.length,ω[0].length]
    ⍔ [ 0 ]
'ρ' ⍙ mcode.shape0

∇ mcode.push0                               // ↓    simple push ω on to stack α  if α is []
    α ⍷ [?] →  α.push(ω)                    //      nb. full ↓ is defined later
    ⍔ α
'↓' ⍙ mcode.push0

∇ mcode.concat0                             // ⍪    simple concat ω to α
    α ⍷ # → α = [ α ]                       //      nb. full ⍪ is defined later
    α ⍷ [] →  α = α.concat(ω)
    ⍔ α
'⍪' ⍙ mcode.concat0

∇ mcode.iota0 : r=[],i=0                    // ⍳    simple generation of vector 0..n
    α ⇋ ⍬ →                                 //      nb. full ⍪ is defined later
        i < ω ⌸ r.push(i++) ; ⍔ r
        ⍔ r
'⍳' ⍙ mcode.iota0
`

r += ⌽.er `
∇ mcode.select : r=[],b                     // ▯    ω[α]
    // ▯ 'select'
    // ▯ α ; ▯ ω ; ▯ ⍷ ω
    // nb. in JS, []==[] is false
    ω.length==0 → ⍔ ⍬
    ω ⍷ # → ω = [ ω ]                       // cvt scalar to vector
    ω ⍷ [?] →                               // ω is [] or [[]]
        α ⍷ # → ⍔ ω[α]                      // if α is scalar then r is ω[α]
        α ⍷ [] → b ⌼ α : r ↓ ω[b] ; ⍔ r     // return vector of ω[α]
    ω ⍷ {} →                                // ω is map
        α!=⍬ →
            δ ⇋ ⍬ → ⍔ ω.get(α)
            ⍔ ω.set(α,δ)
        ⍔ ω
    ω ⍷ . →
        α ⇋ ⍬ →
            ⍔ Object.keys(ω)
        ⍔ ω[α]
    ⍔ 0
'▯' ⍙ mcode.select
`
/*
nb. see Array.copyWithin()
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/copyWithin
nb. selective assignment is written as ω[α] =   or ω[i,j] =
α ▯.v ω  or  α ▯.ω v  ?
▯= cb. for selective assignment as in  [ 1 3 5 ] ▯=.v [ 2 4 6 ]
see:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/with
*/

⌽ `
∇ ⊙.test.select : V,S,P,T
    // ▯ '⊙.test.select'
    V = [ 3 4 5 ]
    // '[3,4]' != ⍕.j
    4 != 1 ▯ V                  → ⍠.↑ '▯ test select 0 failed'
    '[3,4]' != ⍕.j [ 0 1 ] ▯ V     → ⍠.↑ '▯ test select 1 failed'
    S = [['ab','cd'],'ef',['uvw','xyz']]
    '["ef",["ab","cd"],["uvw","xyz"]]' != ⍕.j [ 1 0 2 ] ▯ S → ⍠.↑ '▯ test 2 failed'

    P = [['a',0],['b',1]] ⍙ {}
    'c' ▯.2 P
    2 != 'c' ▯ P                → ⍠.↑ '▯ test select 2 failed'

    T={a:1,b:2}
    2 != 'b' ▯ T                → ⍠.↑ '▯ test select 3 failed'
⊙.test.select 0
`


/*
nb. operators are called as:    mcode.each( α, ω, [op_mod,[f0,mod0]] );
eg.
0 ⊢.a ¨.b 1                     mcode.each( 0, 1, ['b',[mcode.nyi,'a']] );
0 ⊢.⊙.x ¨.⊙.y 1                 mcode.each( 0, 1, [_cp.y,[mcode.nyi,_cp.x]] );
    // nb. ⊙.y contains the modifier value for each ¨
*/

r += '\n// Operator functions'
r += ⍎.er `
∇ mcode.each : b,c,f,fm,i=0,n,r=[],t=[]
    // ▯ 'each' ; ▯ α ; ▯ ω ; ▯ δ ; '⍷ α' ▯ ⍷ α ; '⍷ ω' ▯ ⍷ ω
    [f,fm]=δ[1]
    // ▯ f ; ▯ fm

    ( ω ⍷ . ) ^ α ⍷ [] →
        // each obj
        b ⌼ α
            r ↓ f(b,ω,fm)
        ⍔ r

    ~ α ⍷ [?] → α = [ α ]
    ~ ω ⍷ [?] → ω = [ ω ]

    ω ⍷ [] →
        α ⍷ [] →
            // each α paired with each ω
            n = 0 ▯ ρ α
            b ⌼ ω
                // '  VV α' ▯ α[i%n] ; '  ω' ▯ b
                r ↓ f(α[i++%n],b,fm)
            ⍔ r
        b ⌼ ω
            r ↓ f(α,b,fm)
        ⍔ r
    ω ⍷ [[]] →
        α ⍷ [] →
            // each α paired with each b of ω rows
            n = 0 ⎕ ρ α
            b ⌼ ω
                i = 0
                c ⌼ b : t ↓ f(α[i++%n],c,fm)
                r ↓ t ; t = []
            ⍔ r
    α ⍷ [[]] →
        throw₌ 'error: each on matrix α is NYI'
    ⍔ 0
'¨' ⍙ mcode.each
`


⍎  `
∇ ⊙.test.each : cn,M,S,T
    // ⎕ 'test.each'
    '[4]' != ⍕.j 2 * ¨ 2                                    → ⍠.↑ 'each scalar failed'
    '[4,6]' != ⍕.j 2 * ¨ [ 2 3 ]                            → ⍠.↑ 'each SV failed'
    '[0,1]' != ⍕.j + ¨ [ '0' '1' ]                          → ⍠.↑ 'each + unary failed'
    '[2,4]' != ⍕.j 2 * ¨ [ 1 2 ]                            → ⍠.↑ 'each 0 * failed'
    '[3,5,5]' != ⍕.j [ 1 2 ] { α + ω } ¨ [ 2 3 4 ]          → ⍠.↑ 'each V {} failed'
    cn ← { (α+'')+'x'+(ω+'') }
    '["0x1","0x2"]' != ⍕.j 0 cn ¨ [ 1 2 ]                   → ⍠.↑ 'each foo failed'
    M ← [   0, 1
            2, 3    ]
    '[[1,2],[3,4]]' != ⍕.j 1 +. ¨ M                         → ⍠.↑ 'each matrix failed'
    S ← [   'ab', 'cd'
            'ef', 'gh'  ]
    '[["xb","cd"],["ef","xh"]]' != ⍕.j 'x' { ω.replace(/a|g/g,α) } ¨ S → ⍠.↑ 'each string matrix failed'
    T={a:1,b:2}
    '[2,1]' != ⍕.j [ 'b' 'a' ] ⎕ ¨ T                        → ⍠.↑ 'each object failed'
    ⍔ 0
    try₌
        ⍎.m \`'each 0 ⊢' ⎕ 0 ¨ [ 1 2 ]    // parsing error test (missing fn left of operator)\`
    catch₌
        ⎕ 'error: in test.each'
⊙.test.each 0
`

r += ⍎.er `
```

```
∇ mcode.reduce : b,f,r=null         // ⌿ reduce by applying fn δ over ω  r=α at start
    // ⎕ 'reduce' ; ⎕ α ; ⎕ ω ; ⎕ δ
    [f,fm]=δ[1]
    // ⎕ f ; ⎕ fm

    // f == mcode.max → ⍔ Math.max(...ω)    // todo: optimize?
    // f == mcode.min → ⍔ Math.min(...ω)
    ω ⍷ # → ω = [ ω ]
    ω ⍷ [?] →
        α ⇋ θ →
            b ⌼ ω : r=f(r,b,fm)
            ⍔ r
        b ⌼ ω :
            r ⇋ null →
                r = b ; ⍐
            r=f(r,b,fm)
    ⍔ r
'⌿' ⍙ mcode.reduce
`


⌽ `
∇ ⊙.test.reduce : cn,r,avg
    // ⎕ 'test.reduce'
    // nb. α is initial value and is used to set dyadic call to +
    2 != 0 ⌊ ⌿ [ 2 3 ]                        → ⍠.↑ 'reduce ⌊ failed' // dyadic min of vector
    3 != 0 ⌈ ⌿ [ 2 3 ]                        → ⍠.↑ 'reduce ⌈ failed' // dyadic max of vector
    5 != 0 + ⌿ [ 2 3 ]                        → ⍠.↑ 'reduce + failed'
    avg ← { ( 0 + ⌿ ω ) / ρ ω }
    2.5 != avg [ 2 3 ]                        → ⍠.↑ 'reduce avg failed'
    1 != + ⌿ [ '0' '1' ]                      → ⍠.↑ 'reduce + failed'
    3 != 0 + ⌿ [ 1 2 ]                        → ⍠.↑ 'reduce 0 + failed'
    '1x2' != 0 { (α+'')+'x'+(ω+'') } ⌿ [ 1 2 ]        → ⍠.↑ 'reduce {} failed'
    cn ← { (α+'')+'x'+(ω+'') }
    '1x2' != 0 cn ⌿ [ 1 2 ]                   → ⍠.↑ 'reduce cn failed'
⊙.test.reduce 0
`


/*
r += ⌽.er `
∇ mcode.scan : b,f,i=0,r=0
    ⎕ 'scan' ; ⎕ α ; ⎕ ω ; ⎕ δ
    ⎕ 'scan is not yet implemented'  // zvzv NYI
    ⍔ r
'⍀' ⍙ mcode.scan
`

*/
```

```
r += ⍎.er `
∇ mcode.power : m,f,fm,i=0,r=α
    // ⎕ 'power' ; ⎕ α ; ⎕ ω ; ⎕ δ
    // nb. power function α is r  δ is [mod,i]
    m = δ[0] ; [f,fm]=δ[1]
    // ⎕ m ; ⎕ ⍷ m ; ⎕ f ; ⎕ fm
    ( m ⍷ # ) ^ m>0 →
        i<m ⌸
            // '  i' ⎕ i ; '  mod' ⎕ [fm,i]
            r = f(r,ω,[fm,i]) ; i++
    ( m ⍷ () ) →
        ( i m r ) ^ i<1000 ⌸                // nb. runaway limit
            r = f(r,ω,[fm,i]) ; i++
    ⍔ r
'⍣' ⍙ mcode.power
`


⍎ `
∇ ⊙.test.power
    // ⎕ 'test.power'
    2 != 0 + ⍣.2 1                  → ⍠.↑ 'power 1 failed'
    p2 ← { δ[1] }
    2 != 0 p2 ⍣.3 1                 → ⍠.↑ 'power p2 failed'
    ⊙.c ← { α < 5 }                   // end condition function
    6 != 1 { α + 1 } ⍣.⊙.c 2        → ⍠.↑ 'power p+ failed'
⊙.test.power 0
`


/*
`
∇ mcode.at : m,f,i=0,r
    ⎕ 'at' ; ⎕ α ; ⎕ ω ; ⎕ δ
    ⎕ 'at is not yet implemented'  // zvzv NYI
    ⍔ r
'@' ⍙ mcode.at
`
// ⎕ 'todo: write and test at'
*/

r += ⍎.er `
∇ mcode.rmset
    ⎕ 'rmset not yet implemented'
    ⍔ 0

∇ mcode.factorial
```

```
    ⎕ 'factorial not yet implemented'
    ⍔ 0

∇ mcode.binomial
    ⎕ 'binomial not yet implemented'
    ⍔ 0
`

r += '\n// Vectorized functions'
r += ⍎.er `
∇ mcode.outer : c,r=[]                  // ∘    apply δ over each α paired with each ω
    // outer or cartesian product
    /// notebook outer1 3 tex //Large{ r_{ij} = //delta(//alpha_i,//omega_j)}
```

$$r_{ij} = \delta(\alpha_i, \omega_j)$$

```
    // ⎕ 'outer' ; ⎕ α ; ⎕ ω ; ⎕ δ
    ∇ w : b,s=[]                        // process cols
      b ⌼ ω : s ↓ δ(α,b) ; ⍔ s
    c ⌼ α : r ↓ w(c,ω,δ)                // process rows
    ⍔ r
`


r += ⍎.er `
∇ mcode.vf : f0=null,f1=null
    // at,wt,as,ws

    // vf handles vectorized functions
    // refs:
    //  https://tryapl.org/
    //  https://help.dyalog.com/18.2/index.htm#Language/Primitive%20Operators/Inner%20Product.htm
    //  https://daveremba.com/blogs/os_general/vmath.h  see Matrix * Vector

    // calls to vectorized function handler:
    // op(α,ω,[f0,f1])  where fN = [opN,modN]
    // mcode.vf( α, ω, [ f0, f1 ] )

    // ⎕ 'mcode.vf' ; ⎕ α ; ⎕ ω
    // 'δ0' ⎕ δ[0] ; 'δ1' ⎕ δ[1]

    δ[0] != θ → f0 = δ[0,0];   // set vectorized functions
    δ[1] != θ → f1 = δ[1,0];
    // f1 ⇝ θ → f1 ← { ω } // test
    // ⎕ f0 ; ⎕ f1

    // special cases using array spread
    f0=='norm'      → ⍔ α*Math.hypot(...ω)
```

```
f0=='atan2'      → ⍔ Math.atan2(...ω)

α ⍷ # → α = [ α ]             // scalar to vector promotion
ω ⍷ # → ω = [ ω ]

// outer product handler
f0=='outer' → ⍔ mcode.outer(α,ω,f1)

// nb. inner product α f0.f1 ω expands to f0 ⌿ α f1 ¨ ω
// dot product is a special case +.×
// APL: f0 / α f1 ¨ ω     mcode: f0 ⌿ α f1 ¨ ω

// define handlers

∇ MM : i=0,j=0,k=0,t,u
    // α and ω are matrices, and u is size i,j of result matrix t
    // ⎕ 'MM'
    u = ( 0 ⌷ ⍴ α ) ⍮ 1 ⌷ ⍴ ω ;
    t = u ⍙ [[]]
    // ⎕ u ; ⎕ t
    // if f0 = null then we have element by element operation:
    /// notebook vfMM1 3 tex //Large{t_{ij} = f_1({//alpha_{ij},//omega_{ij})}}
```

$$t_{ij} = f_1(\alpha_{ij}, \omega_{ij})$$

```
    f1 ↯ θ →
        i ⌼ ⍳ 0 ⌷ u : j ⌼ ⍳ 1 ⌷ u
            t[i,j] = α[i,j] f0 ω[i,j]
        ⍔ t

    // if f0 = sum and f1 = times then we have matrix multiply:
    /// notebook vfMM2 4 tex //Large{t_{ij} = //sum_{k} {//alpha_{ik} //times //omega_{kj}}}
```

$$t_{ij} = \sum_k \alpha_{ik} \times \omega_{kj}$$

```
    // the general case is:
    /// notebook vfMM3 3 tex //Large{t_{ij} = f_0(t_{ij},f_1(//alpha_{ik},//omega_{kj}))}
```

$$t_{ij} = f_0(t_{ij}, f_1(\alpha_{ik}, \omega_{kj}))$$

```
    i ⌼ ⍳ 0 ⌷ u : j ⌼ ⍳ 1 ⌷ u : k ⌼ ⍳ 1 ⌷ ⍴ α
        t[i,j] = t[i,j] f0 α[i,k] f1 ω[k,j]
    ⍔ t
∇ MV : i=0,j=0,t,u,v
    f1 ↯ θ →
        u = ⍴ α ; v = 0 ⌷ ⍴ ω ; t = u ⍙ [[]]
        i ⌼ ⍳ 0 ⌷ u : j ⌼ ⍳ 1 ⌷ u
            t[i,j] = α[i,j] f0 ω[j%v]
```

```
      ⍔ t
    u = 0 ⌷ ⍴ ⍺ ; v = ⍴ ⍵ ; t = u ⍙ []
    i ⌼ ⍳ u : j ⌼ ⍳ 1 ⌷ ⍴ ⍺
      t[i] = t[i] f0 ⍺[i,j] f1 ⍵[j%v]
    ⍔ t
  ∇ VM : i=0,j=0,t,u,v
    f1 ↯ θ →
      u = ⍴ ⍵ ; v = 0 ⌷ ⍴ ⍺ ; t = u ⍙ [[]]
      i ⌼ ⍳ 0 ⌷ u : j ⌼ ⍳ 1 ⌷ u
        t[i,j] = ⍺[i%v] f0 ⍵[i,j]
      ⍔ t
    u = 0 ⌷ ⍴ ⍵ ; v = ⍴ ⍺ ; t = u ⍙ []
    i ⌼ ⍳ u : j ⌼ ⍳ 1 ⌷ ⍴ ⍵
      t[j] = t[j] f0 ⍺[i%v] f1 ⍵[i,j]
    ⍔ t
  ∇ VV : i=0,j=0,t,u,v,n,r=0
    max ← { Math.max(⍺,⍵) }
    n = ( ⍴ ⍵ ) max ( ⍴ ⍺ ) ; u = ⍴ ⍺ ; v = ⍴ ⍵ ; t = n ⍙ []
    f1 ↯ θ →
      i ⌼ ⍳ n : t[i] = ⍺[i%u] f0 ⍵[i%v]
      ⍔ t
    i ⌼ ⍳ n : r = r f0 ⍺[i%u] f1 ⍵[i%v]
    ⍔ r

  // dispatch handlers
  ⍵ ⍷ [[]] →
    ⍺ ⍷ [[]] → ⍔ ⍺ MM ⍵
    ⍺ ⍷ [] → ⍔ ⍺ VM ⍵
    ⍔ 0
  ( ⍵ ⍷ [] ) ^ ⍺ ⍷ [[]] → ⍔ ⍺ MV ⍵
  ( ⍵ ⍷ [] ) ^ ⍺ ⍷ [] → ⍔ ⍺ VV ⍵
  ⍵ ⍷ [] → ⍔ [] VV ⍵

  ⍔ 0
`

r += '\n// General data and assert functions'
r += ⍎.er `
mcode.assertOpts='';
∇ mcode.assert : r,b,d='assert',e,s   // ?=   assert ⍺ = 'code result'
  // δ :  x  use ⍵ as result only, do not execute ⍎ ⍵
  //      d  show debug messages
  //      t  do not throw on errors
  // uses built-in != with JSON rather than vectorized comparison
  ∇ opts : X,Y
    ⍔ (mcode.assertOpts.indexOf(⍵)>=0) ∨ b.indexOf(⍵)>=0
  b = δ==null?'':δ
  e = opts 'd'
  // e →
  //     ⎕ 'assert' ; ⎕ α ; ⎕ ω ; ⎕ δ
  opts 'x' → r = ⍕.j ω
  ⋄
      r = ⍕.j ⍎ ω ; d += ' of ' + ω   // add expr tested to d
  α != r →
      s = d+' failed: ' + α + ' ?= ' + ω + ', got '+r
    opts 't' → '' ⎕ s
    ⋄ ⍠.↑ s
  ⋄ e → ⎕ d+' passed: ' + α + ' == ' + ω
  ⍔ r
'?=' ⍙ mcode.assert
`


⍎ `
∇ ⊙.test.format
  '[3,0]' ?= '⍕ [ π 0.1234 ]'
  '[3.142,0.123]' ?= '3 ⍕ [ π 0.1234 ]'   // nb. JSON result is a string
⊙.test.format 0
`


⍎ `
∇ ⊙.test.vf0
  '[1,3]' ?= '¦ [ 1.1 2.9 ]'
  '[2.236]' ?= '3 ⍕ 1 ¦ [ 1 2 ]'
  '[0.464]' ?= '3 ⍕ atan2θ [ 1 2 ]'
  '[2,5]' ?= '⌈ [ 1.1 4.8 ]'
  '[3,4]' ?= '[ 3 2 ] ⌈ [ 1 4 ]'
  '[3]' ?= '1 +. 2'
  '[2,3]' ?= '[ 1 2 ] +. 1'
  '[2,3]' ?= '1 +. [ 1 2 ]'
  '[2,4]' ?= '[ 0 1 ] +. [ 2 3 ]'
  '5' ?= '1 +.× [ 2 3 ]'
  ⍔ 0
⊙.test.vf0 0
`


⍎ `
∇ ⊙.test.vf1 : M,N,V
  // ⎕ 'test.vf1'
  // ⍠.a 'd'
  // nb. M,N,V are locals instead of in ⊙, so we use ?=.x not ?=
  //  (no execute form)
  // nb. M f.g N is f / g on M cols and N rows

  M ← [   1,  2
          3,  4   ]
  N ← [   5,  6
          7,  8   ]
  V ← [   9,  10  ]
  // ⎕ M ; ⎕ N ; ⎕ V

  '[[2,3],[4,5]]' ?=.x 1 +. M
  '[[2,3],[4,5]]' ?=.x M +. 1
  '[[2,4],[6,8]]' ?=.x M +. M

  // inner product tests
  '[[19,22],[43,50]]' ?=.x M +.× N
  '[[23,34],[31,46]]' ?=.x N +.× M
  '[29,67]' ?=.x M +.× V
  '[39,58]' ?=.x V +.× M
  '[12,18]' ?=.x 3 +.× M
  '[9,21]'  ?=.x M +.× 3

  /* APL verification:
        M ← 2 2 ρ ι 4
        M
  1 2
  3 4
        N ← M + 4
        N
  5 6
  7 8
        V ← 9 10
        V
  9 10
        M +.× N
  19 22
  43 50
        N +.× M
  23 34
  31 46
        M +.× V
  29 67
        V +.× M
  39 58
  */

  ⍔ 0
```

```
⊙.test.vf1 0
`

⍎ `
∇ ⊙.test.vf2
    // ⎕ 'vectorized functions'
    // ⍠.a 'd'
    '[2,4]' ?= '⌈ [ 1.2 3.4 ]'
    '[[0,0],[2,3]]' ?= '[[0],[1]] +.× [[2,3]]'
    // nb. vectors are treated as column vectors when appropriate, without transposition
    '[0,2]' ?= '[[0],[1]] +.× [2,3]'
    '3' ?= '[0,1] +.× [2,3]'
    '[3,0]' ?= '[ 2 3 ] +.× [[0],[1]]'
    '3' ?= '[ 2 3 ] +.× [ 0 1 ]'
    '[2,4]' ?= '[ 0 1 ] +. [ 2 3 ]'
    '[3,4]' ?= '1 +. [ 2 3 ]'
    '[2]' ?= '1 +. 1'
    '[2,3]' ?= '2 ⌈ ⌈ [ 0.1 2.7 ]'
    '[7.07107]' ?= '5 ⍕ 1 ¦ [ 3 4 5 ]'
    '[0,1]' ?= 'sinθ [ 0 π/2.0 ]'
    '[1,2]' ?= '1 ⌈ [ 0 2 ]'
    '[0,2]' ?= '⌈ [ 0 2 ]'
    '[2,3]' ?= '2 ⌈ ⌈ [ 0.1 2.7 ]'
⊙.test.vf2 0
`

⍎ `
∇ ⊙.test.matrix : t,a
    t = π ÷ 2
    '[1]' ?= 'sinθ 0.5 × π'
    '[1]' ?= 'sinθ π ÷ 2'
    '[1.571]' ?= '3 ⍕ atan2θ [ 1 0 ]'
    ⊙.Mz ← [    cosθ t,     - sinθ t,       0,  // rotate on Z axis
                sinθ t,     cosθ t,         0,
                0,           0,             1   ]
    // 'Mz' ⎕ ⊙.Mz
    '-1' ?= '⊙.Mz[0,1]'
    ⍫ ⊙.Mz        // delete
⊙.test.matrix 0
`

⍎ `
∇ ⊙.test.outer
    // ⎕ 'test.outer'
    '[[1]]' ?= '0 ∘.+ 1'
    '[[4,5],[5,6]]' ?= '[ 1 2 ] ∘.+ [ 3 4 ]'
    '[[3,4],[6,8]]' ?= '[ 1 2 ] ∘.× [ 3 4 ]'
    ⍔ 0
⊙.test.outer 0
`


/* NIU each Map
r += ⍎.er `
∇ mcode.each : k,v,r=[]                    // ¨    apply α over ω   uses vf
    ω ⍷ {} →
        [k,v] ⌼ ω : r.push( k α v )       //  map
        ⍔ r
    ⍔ mcode.vf(δ,ω,[α,null])
'¨' ⍙ mcode.each
`


⍎ `
∇ ⊙.test.eachMap : U,V,M,r=[]
    V = [ 0 1 2 ]
    U = [ 2 3 ]
    // ⎕ V ; ⎕ U
    '[[0,2],[1,3],[2,2]]' ?=.x ⍕.j V { [α,ω] } ¨ U     // .x since U,V are locals

    M ← [   'k1',   0,
            'k2',   1   ]
    // ⎕ M
    Mp = M ⍙ {}                            // create map
    // ⎕ Mp
    { r.push( [ α , ω ] ) } ¨ Mp           // iterate over map
    r = ⍕.j r
    '[["k1",0],["k2",1]]' ?=.x r           // check
⊙.test.eachMap 0
`
*/

/* NIU zvzv
r += ⍎.er `
∇ mcode.notEqual : r=1,i=0             // ≠    hybrid function for arrays  strings  scalars
    // ⎕ 'notEqual' ; ⎕ α instanceof‗ Array ; ⎕ ω instanceof‗ Array
    ( α instanceof‗ Array ) ^ ( ω instanceof‗ Array ) ^ α.length==ω.length →
        i < α.length ⌸ α[i]!=ω[i++] → ⍗
        r = 0
    ⋄ r = Number(α!=ω)              // nb. cast result to number
    ⍔ r
'≠' ⍙ mcode.notEqual
`
```

```
r += ⌽.er `
∇ mcode.equal                       // =    calls ≠
    ⍔ Number( ! α ≠ ω )
'≈' ⍙ mcode.equal
*/

⌽ `
∇ ⊙.test.equals
    ⎕ '⊙.test.equals'
    // equality, matrix, and assert tests
    // 1 ?=.x 1                    // assert test (no mexec)
    1 ?= '1'                    // assert test
    ⊙.M ← ⍙ '[[]]'              // create test  use ⊙ so assert can do mexec
    1 ?= \`'[[]]' ≈ ⍷ ⊙.M \`
    0 ?= \`'[[]]' ≠ ⍷ ⊙.M\`
    '[[]]' ?= '⍷ ⊙.M'
    ⊙.M ← [ 1, 0, 0,            // also does create and initializes
            0, 1, 0,
            0, 0, 1, ]
    ⊙.N ← [ 0, 1, 2 ]
    '[[]]' ?= '⍷ ⊙.M'
    '[]' ?= '⍷ ⊙.N'
    ⍫ ⊙.M ;  ⍫ ⊙.N
    // ⎕ ⊙
// ⊙.test.equals 0
`


/*
r += ⌽.er `
∇ mcode.theta : r
    // NIU - replaced, performance better as direct calls in OPsub()
    // was: implements trig fns and also tests switch/case
    α ≈ 'atan2' → r = Math.atan2(ω,δ)           // 'atan2' θ.x y
    ◇ ⌺.s α                                     // switch on fn name
        ⌺ 'sin'   : r = Math.sin(ω)   ; ⍗
        ⌺ 'cos'   : r = Math.cos(ω)   ; ⍗
        ⌺ 'tan'   : r = Math.tan(ω)   ; ⍗
        ⌺ 'sinh'  : r = Math.sinh(ω)  ; ⍗
        ⌺ 'cosh'  : r = Math.cosh(ω)  ; ⍗
        ⌺ 'tanh'  : r = Math.tanh(ω)  ; ⍗
        ⌺ 'asin'  : r = Math.asin(ω)  ; ⍗
        ⌺ 'acos'  : r = Math.acos(ω)  ; ⍗
        ⌺ 'atan'  : r = Math.atan(ω)  ; ⍗
        ⌺ 'asinh' : r = Math.asinh(ω) ; ⍗
        ⌺ 'acosh' : r = Math.acosh(ω) ; ⍗
        ⌺ 'atanh' : r = Math.atanh(ω) ; ⍗
        ⌺.d       : r = Math.PI                 // default is pi, NIU: use π symbol
    ⍔ r
'θ' ⍙.L 'theta'                                 // Left arg is unquoted
`

*/

r += ⌽.er `
∇ mcode.concatenate : r=θ,i=0,e,t               // ⍪  ravel or concatenate
    // ⎕ 'concatenate' ; ⎕ α ; ⎕ ω ; ⎕ δ

    α ⇋ θ →                                     // monadic: ravel

        ω ⍷ [?] →
            ω[0] ⍷ ''    → ⍔ ω.join('')         // join as string, no delimiter
            1 == ρ ω    → ⍔ ω[0]                // [ x ] to x
            ⍔ ω.flat()                          // like APL's enlist, make depth 1

    δ == θ →                                    // default is concat/join first (row) axis

        α ⍷ '' → ⍔ ω.join(α)                    // join as string

        ( α.concat ⍷ () ) ^ ω.concat ⍷ () →     // concat if both are arrays or matrices
            ⍔ α.concat(ω)

    ⋄ ( δ == 'r' )  →                           // laminate first (rows) axis
        α ⍷ [[]] →                              // α is matrix, append ω to last col of α
            r = 11  // todo: NYI
        ⋄
            r = [α] ; r.push(ω)
    ⋄ ( δ == 'c' )  →                           // laminate second (columns) axis
        α ⍷ [[]] →                              // α is matrix, append ω to last col of α
            r = []
            e ⌻ α : r.push(e.concat(ω[i++]))
        ⋄
            r = α.concat(ω)
    ⍔ r
    // todo: catenate on strings? (but immutable), table (row or col)
'⍪' ⍙ mcode.concatenate
`


⌽ `
∇ ⊙.test.concatenate
    // ⎕ 'test.concatenate'
    // ⍠.a 'd'
    '"ab"'  ?= \` ⍪ [ 'a' 'b' ]\`
    '"axb"' ?= \` 'x' ⍪ [ 'a' 'b' ]\`
    ⊙.M ← [ 1, 2,
             3, 4 ]
    ⊙.V ← [ 0, 1, 2 ]
    // ⎕ ⊙.V ; ⎕ ⊙.M
    // 'r' ⎕ ⊙.V ⍪.r ⊙.V
    ⊙.W = [[0,1],2,3]
    '[0,1,2,3]' ?= '⍪ ⊙.W'                  // ravel
    '["ab","cd","ef","gh"]' ?= "⍪ [['ab','cd'],['ef','gh']]"    // ravel string matrix

    '[0,1,2,0,1,2]' ?= '⊙.V ⍪ ⊙.V'          // concat/join vectors
    '[[0,1,2],[0,1,2]]'  ?= '⊙.V ⍪.r ⊙.V'   // laminate vector to make matrix
    '[[1,2],[3,4],0,1,2]' ?= '⊙.M ⍪ ⊙.V'    // concat matrix to vector
    '[0,1,2,[1,2],[3,4]]' ?= '⊙.V ⍪ ⊙.M'    // concat vector to matrix
    '[[1,2,0],[3,4,1]]' ?= '⊙.M ⍪.c ⊙.V'    // laminate cols of matrix with vector
    '[0,1,2,[1,2],[3,4]]' ?= '⊙.V ⍪.c ⊙.M'  // laminate vector to matrix
⊙.test.concatenate 0
`


r += Φ̲.er `
∇ mcode.iota : r=[],i=0,b,c                 // ⍳     generate   or   where
    // ⎕ 'iota' ; ⎕ α ; ⎕ ω
    α ⇋ θ →                                 // monadic: generate vector of 0..n
        i < ω ⌸ r.push(i++) ; ⍔ r
        ⍔ r
//    ( α ⍷ [] ) ^ ω ⍷ [] →                   // indices of α in ω [ ]  r is [ ]
//        b ⌼ α : c ⌼ ω :
//            b == c → r.push(c) ◇ r.push(null);
//        ⍔ r
    ( α ⍷ [] ) ^ ω ⍷ {} →                   // values of ω when key α found in ω  r is [ ]
        b ⌼ α :
            c=ω.get(b) ; r.push(c??null)
        ⍔ r
    α ⍷ '' →
        ! ω ⍷ '' → ⍔ -1                     // α not in ω
        ⍔ ω.indexOf(α)                      // string α in string ω
    // NIU - use α ⍳ ¨ ω
    // ( α ⍷ '' ) ^ ω ⍷ [] →                 // string α in each string ω
    //     b ⌼ ω :
    //         b ⍷ '' →
    //             r ↓ b.indexOf(α)
    //         ◇ r ↓ -1
    //     ⍔ r
    ⍔ 0
'⍳' ⍙ mcode.iota
`
```

```
⌽ `
∇ ⊙.test.iota
    // ⎕ 'test.iota'
    '1' ?= 'x' ⍳ 'axb'
    ⊙.M ← [ 0, 'a',
            1, 'b' ]
    ⊙.m = ⊙.M ⍙ {}
    // ⎕ ⊙.m ; 't' ⎕ [ 0 2 ] ⍳ ⊙.m
    '["a",null]' ?= '[ 0 2 ] ⍳ ⊙.m'    // search map
    '[1,-1]' ?= \`'ab' ⍳ ¨ [ 'xab' 'cde' ]\`  // search each string
⊙.test.iota 0
`

r += ⌽.er `
∇ mcode.shape : r=[],b,c,i=0,j              // ρ    length or size of ω
    //                                      or  reshape vector ω by vector α
    // ⎕ 'shape' ; ⎕ α ; ⎕ ω ; ⎕ ⍷ ω
    α ⇌ θ →
        ω ⍷ [[]] → r=[ω.length,ω[0].length] // assumes matrix not ragged
        ⋄ ω ⍷ [] → r=[ω.length]
        ⋄ ω ⍷ {} → r=ω.size                 // nb. non-vector result for maps
        ⋄ ω ⍷ # → r=0                       // shape of scalar and rank 0
        ⋄ ω ⍷ '' → ⍔ ω.length               // string
        ⋄ r = null                          // non-array type
        ⍔ r

    ω ⍷ # → ω = [ ω ]                       // convert scalar ω to vector
    ω ⍷ [[]] → ω = ω.flat()                 // convert array to vector
    ! ( ω ⍷ [] ) → ⍔ r
    ⋄ ω.length ⇌ 0 → ⍔ r                    // can't reshape nothing
    ⋄ α ⍷ [] →                              // reshape vector ω into matrix
        α.length < 2 → α = [ α 1 ]
        i < α[0] ⌸                          // i : row  j : ω index  c : col
            b = [] ; c=0 ; c < α[1] ⌸
                j = (i*α[1]+c)%ω.length
                b=b.concat(ω[j]) ; c++
            r.push(b) ; i++
    ⋄ α ⍷ # →                               // reshape vector ω into new vector
        i < α ⌸
            j=i%(ω.length) ; r.push(ω[j]); i++
    ⍔ r
'ρ' ⍙ mcode.shape
`

⌽ `
∇ ⊙.test.shape
```

```
    // ⎕ 'test.shape'
    ⊙.M ← [ 1, 2,
            3, 4 ]
    ⊙.V ← [ 0, 1, 2 ]
    '3' ?= \`ρ 'abc'\`
    ⊙.M = ⊙.V ⍪.r ⊙.V
    // ⎕ ⊙.V ; ⎕ ⊙.M ;
    // 'shape V' ⎕ ρ ⊙.V
    '[3]' ?= 'ρ ⊙.V'
    // 'shape M' ⎕ ρ ⊙.M
    '[2,3]' ?= 'ρ ⊙.M'
    // 'rank V' ⎕ ρ ρ ⊙.V
    '[1]' ?= 'ρ ρ ⊙.V'
    // 'rank M' ⎕ ρ ρ ⊙.M
    '[2]' ?= 'ρ ρ ⊙.M'
    '[[0,1,2],[3,4,5]]' ?= '[ 2 3 ] ρ ι 6'
    // 'reshape' ⎕ 4 ρ ι 2
    '[0,1,0,1]' ?= '4 ρ ι 2'
    // 'reshape 0 0 0 ' ⎕ 3 ρ 0
    '[0,0,0]' ?= '3 ρ 0'
    // '3x3 identity' ⎕ [ 3 3 ] ρ [ 1 0 0 0 ]
    '[[1,0,0],[0,1,0],[0,0,1]]' ?= '[ 3 3 ] ρ [ 1 0 0 0 ]'
⊙.test.shape 0
`

r += ⌽̲.er `
▽ mcode.push                                // ↓    push ω on to stack α  or 'drop' α elements from ω
    // ⎕ 'push' ; ⎕ α ; ⎕ ω
    α ⇋ null → ⎕ 'monadic ↓ not impl\\n'
    ( α ⍷ [] ) ∨ α ⍷ [[]] → α.push(ω)
    ◊ ( ω ⍷ [] ) ∨ ( ω ⍷ '' ) →             // drop  α is not [] or [[]]
        α > 0 → ⍔ ω.slice(α)                // nb. slice does -not- modify strings, but does modify arrays
        ◊ ⍔ ω.slice(0,α)
    ⍔ α
    // ?  zvzv
    // ◊ α ⍷ [[]] →
    //     α.splice(α.length,0,ω) ; ⍔ α
    // ◊ α ⍷ [] →
    //     α.push(ω) ; ⍔ α
'↓' ⍙ mcode.push

▽ mcode.pop : r                             // ↑    pop from stack ω  or 'take' α elements from ω
    α ⇋ null →
        r = ω.pop() ; ⍔ r
    ◊ ( ω ⍷ [] ) ∨ ( ω ⍷ '' ) →             // take
        α > 0 → ⍔ ω.slice(0,α)              // nb. slice does -not- modify strings, but does modify arrays
        ⋄ ⍔ ω.slice(α)
    ⍔ ω
'↑' ⍙ mcode.pop
`

⌽ `
∇ ⊙.test.push
    // ⎕ 'test.push'
    // ⍠.a 'd'
    ⊙.v = [ 'a', 'b' ]
    '["a","b","bks","del"]' ?= "( ⊙.v ↓ 'bks' ) ↓ 'del'"
    // '[0,1,2]' ?= '⍳ 3'
    ⊙.v = ⍳ 3
    '[0,1]' ?= '2 ↑ ⊙.v'
    '[2]' ?= '2 ↓ ⊙.v'
    ⊙.s = 'abc'
    '"ab"' ?= '2 ↑ ⊙.s'
    '"c"' ?= '-1 ↑ ⊙.s'
    '"a"' ?= '1 ↑ 2 ↑ ⊙.s'
    '"c"' ?= '2 ↓ ⊙.s'
    '"ab"' ?= '-1 ↓ ⊙.s'
⊙.test.push 0
`

r += ⌽.er `
∇ mcode.insert                              // »    insert (at head or at index)
    // ⎕ 'insert' ; ⎕ α ; ⎕ ω ; ⎕ δ
    ω ⍷ [] →
        δ ↯ θ → ω.unshift(α)
        δ ⍷ # → ω.splice(δ,0,α)
    ⋄ ω ⍷ '' →
        δ ↯ θ → ⍔ α + ω
        δ ⍷ # → ⍔ ω.slice(0,δ)+α+ω.slice(δ)
        // nb. return by value, since strings are immutable
    ⍔ ω
    // todo: insert at index
'»' ⍙ mcode.insert

∇ mcode.remove
    ω ⍷ [] → ω.shift(α)
    // todo: ⋄ ω ⍷ '' → ⍔ α + ω                // «    remove (from head or at index)
    ⍔ ω
'«' ⍙ mcode.remove
`

⌽ `
```

```
▿ ⊙.test.insert
    // ⎕ 'test.insert'
    ⊙.s = 'abc' ; ⊙.v = ⍳ 3
    '[0,"xyz",1,2]' ?= '"xyz" ».1 [ 0 1 2 ]'
    '"xabc"' ?= '"x" » ⊙.s'
    '[[8,9],0,1,2]' ?= '[ 8 9 ] » ⊙.v'
    ⊙.v = ⍳ 3
    '["x",0,1,2]' ?= '"x" » ⊙.v'
    '"abcdef"' ?= '"abc" » "def"'
    '"dabcef"' ?= '"abc" ».1 "def"'
⊙.test.insert 0
`

r += Φ̲.er `
▿ mcode.match : i                       // string search  δ : e exec
    α ⍷ '/' →                           // α is a regexp
        δ==θ → ⍗ 0+α.test(ω)
        δ=='e' → ⍗ α.exec(ω)            // returns exec() array
        // δ=='m' → ⍗ α.matchAll(ω)     // returns matchAll() array  m flag required on regexp
    ⋄ α ⍷ '' →
        i=ω.indexOf(α) ; ⍗ 0+i>=0       // simple string search
'≡' ⍙ mcode.match

▿ mcode.replace                         // string replace   α regexp  δ replacement
    // ⎕ 'replace' ; ⎕ α ; ⎕ ω ; ⎕ δ
    ω ⍷ '' →
        α ⍷ '/' →                       // α is a regexp
            α.global →
                ⍗ ω.replaceAll(α,δ)
            ⍗ ω.replace(α,δ)
        ⍗ ω.replaceAll(α,δ)
    ⍗ 0
'≢' ⍙ mcode.replace
`

Φ̲ `
▿ ⊙.test.string : e={}
    // ⎕ 'test.string'
    ⊙.s = 'abc ok xyz abc'
    // 1 ?= '/ok/ ≡ ⊙.s'
    // '"abc nak xyz abc"' ?= '/ok/ ≢.nak ⊙.s'
    // '"def ok xyz def"'  ?= '/abc/g ≢.def ⊙.s'
    '"abcSokSxyzSabc"' ?=.x /\\s/g ≢.S ⊙.s
    '"abcSokSxyzSabc"' ?= '\/\\\\s\/g ≢.S ⊙.s'
    e.sl = 'L'
    '["L op y","L op2 y"]' ?=.x /x/g ≢.e.sl ¨ [ 'x op y' 'x op2 y' ]
```

```
⊙.test.string 0
`

r += ⏀.er `
∇ mcode.split                     // ⊃    split
  // ▯ 'split' ; ▯ α ; ▯ ω ; ▯ δ
  ω ⍷ '' → ⍔ ω.split(α)
  ⍔ ω
'⊃' ⍙ mcode.split

∇ mcode.join                      // ⊂    join
  // ▯ 'join' ; ▯ α ; ▯ ω ; ▯ δ ; 't' ▯ ⍷ ω
  ω ⍷ [] → ⍔ ω.join(α)
  ⍔ ω
'⊂' ⍙ mcode.join
`

⏀ `
∇ ⊙.test.splitjoin
  '["abc","def"]' ?= "'x' ⊃ 'abcxdef'"
  '"abc;def"' ?= '";" ⊂ ["abc","def"]'
⊙.test.splitjoin 0
`

r += ⏀.er `
∇ mcode.memberof : r=[],b         // ∈    is α member of set ω  α and ω may be vectors
  ω ⍷ {} → b ⌼ α : r ↓ ω.has(b)
  ⋄ ω ⍷ [] →
    α ⍷ [] → b ⌼ α : r ↓ 0+ω.includes(b)
    ⋄ r=0+ω.includes(α)
  ⋄ ω ⍷ '' → b ⌼ α : r ↓ 0+(ω.indexOf(b)>-1)
  ⍔ r
'∈' ⍙ mcode.memberof
`

⏀ `
∇ ⊙.test.memberof
  // ∈ test
  [ 1 0 ] ?= '[ 2 3 ] ∈ [ 0 1 2 ]'
// ⊙.test.memberof 0
`

r += ⏀.er `
∇ mcode.sort : d,rev=1        // ⍋   sort list ω  α is a sort function
  // δ flags:   r reverse sort direction, v return value array else index array
  // ▯ 'sort' ; ▯ α ; ▯ ω ; ▯ δ
```

```
    ω ⍷ # → ω = [ ω ]
    ! ω ⍷ [] → ⍔ 0                  // ω must be a list
    d = ω.map((e,i)=>[i,e])         // build data array
    0 ≤ 'd' ⍳ δ → rev=-1            // reverse sort direction
    ∇ comp
       // ▯ 'comp' ; ▯ α ; ▯ ω ; ▯ d
       α[1] > ω[1] → ⍔ rev
       α[1] < ω[1] → ⍔ -1*rev
       ⍔ 0
    α ⇋ θ → α ← comp                // set comparator fn
    d.sort(α)                       // sort data array in place
    // 'd sorted' ▯ d
    0 ≤ 'v' ⍳ δ →                   // return values, not indices
       ⍔ d.map(v=>v[1])
    ⍔ d.map(v=>v[0])                // return index array
'⍋' ⍙ mcode.sort

∇ mcode.sortDown
    ⍔ mcode.sort(α,ω,'r'+(δ??''))
'⍒' ⍙ mcode.sortDown

`
// see
// https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort#sorting_with_map

⍎ `
∇ ⊙.test.sort
    // ▯ 'test.sort'
    ⊙.V = [ 'ab' 'xy' 'cd' ]
    '[0,2,1]' ?= '⊙.a = ⍋ ⊙.V'
    '["ab","cd","xy"]' ?= '⊙.a ▯ ⊙.V'
    '["ab","cd","xy"]' ?= '⍒.v ⊙.V'
⊙.test.sort 0
`


// r += ⍎.er `
// ∇ mcode.quat                                          // matrix rotations and quaternion operations
//     ⍔ 0 // NYI
// '○' ⍙ 'mcode.quat'      // or θ ?
// `

// todo: table functions
/*
    tables are objects similar to a database table
    format ⍕ converts tables to JSON or a readable text format
    create ⍙ creates new tables
```

```
*/

/* NIU
r += ⍜.er `
∇ mcode.helpFn
    ω=='more' →
        ⍠ 0
    ⍔ ''

∇ mcode.roll
    δ=='help' → mcode.help()
    ◇ ω ⍷ '' → mcode.helpFn ω
    // roll or deal
    ⍔ θ
'?' ⍙ mcode.roll
`
*/

r += '\n// File read/write functions'
r += ⍜.er `
∇ mcode.read : p,f      // read something or load URL  δ is operation  α callback optional or returns promise
    // ⎕ 'mcode.read'
    // if α is null then  1) caller is async fn  2) mexec inserts await left of ⍃

    δ ϟ θ →         ⍔ mcode.getURL(α,ω)                 // read using fetch from server

    ◇ δ=='load' →   ⍔ mcode.loadURL(α,ω)                // uses DOM createElement to load URL
    ◇ δ=='ls'   →   ⍔ mcode.getURL(α,'io.php?op=LS')    // file list
    ◇ δ=='ll'   →   ⍔ mcode.getURL(α,'io.php?op=LL')    // file list with details: ls -AgGhFR

    ◇ δ=='t' →                                          // make async function wait
        ⎕ 'read timer ' + ω + 'ms'
        ⍔ new_Promise(rs=>setTimeout(()=>rs(0),ω))      // async caller will wait
    ◇ δ=='p' →                                          // must be used in ∇.a  async fn
        p = new_Promise(rs=>f=rs)                       // save the resolver function
        ω instanceof_ Function → ω f    // fn ω calls its ω arg which is rs when done
        ⍔ p
    ◇ δ=='c' →              // read console/stdin, must be used in ∇.a  async fn
        // ⎕ 'mcode.getInput'
        ⎕.nnl ω
        ⍔ mcode.getInput 0  // returns promise to get console input (see mcode_ide)

    ⍔ ω

'⍃' ⍙ mcode.read            // nb. await is inserted by transpiler
`
```

```
⌽ `
∇.a ⊙.test.futures : p      // async function
    ∇.a f                   // function that will complete in the future
        ⎕ 'ok'
        ⍃.t 1000            // await for timer in ms  nb. IDE did not wait
        ω('done')           // ω is resolver fn
    p = ⍃.p f               // await inserted, f will return result in the future
    ⎕ p
    p = ⍃.t 1000            // await for timer in ms  nb. IDE did not wait
    ⎕ p
// ⊙.test.futures 0         // nb. test is NIU to not delay startup
`


r += ⌽.er `
∇.a mcode.write : p   // write to server  restrictions on server side  α callback optional or returns promise
    // ⎕ 'write' ; ⎕ α ; ⎕ δ
    p = await‗fetch('io.php?op=PUT&fn='+δ+'&auth='+mcode.serverAuth,‗
        method:"POST",headers:{"Content-Type":"application/octet-stream"},body:ω‗
        // nb. last ‗ suppresses ;
    )
    α instanceof‗ Function →
        !p.ok → α(p.status,'')
        ⋄ p.text().then(ω=>α(p.status,ω))              // used with callback fn
    ⋄
        p.ok → ⍔ p.text()                               // used with async/await or .then
        ⋄ ⍔ p.text()
    // ).then(rsp=>rsp.text()).then(data=>α(0,data))    // .then callback style (for ref)
    ⍔ null
'⍄' ⍙ mcode.write   // nb. await is inserted by transpiler
`


⌽ `
∇.a ⊙.test.io0 : seq=1
    ⍠.busy 0
    p = await‗ ⍠.timer ω                        // await for timer in ms here
    ⎕ 'timer done'
    seq →                                       // run sequentially
        ⊙.test.io1 0
    ⍠.done 0
    ⍔ 0
`


⌽ `
∇.a ⊙.test.io1 : f,g1,g2,rs
```

```
    ⍠.busy 0

    // { ⎕ ω } ⍄.'test.txt' 'start\\nline 1\\nline 2\\nend'         // write test
    // f = ⍄.'test.txt' 'start\\nline 1\\nline 2\\nend'             // async write test
    // ⎕ f

    // { ⎕ 'getURL: status = '+α+' read:\\n'+ω } ⍃ 'out/test.txt'   // read test with callback
    // g = ⍃ 'out/test.txt'                     // async read test  nb. ⍃ does await

    g1 = ⍃.ls ''                                // async tests
    g2 = ⍃.ll ''
    ⎕ 'io tests'
    ⎕ 'ls:\\n'+g1                               // debug
    // ⎕ 'll:\\n'+g2                            // debug
    // { ⌽.m 'test' ⎕ ω } ⍃ 'tests.mc'          // run more mcode tests
    ⎕ 'test.io done'
    // ⍠.done 1                                 // 1 returned as result to await
    // ⍠.done ⍃ 'out/test.txt'                  // or return file as result directly

▽ ⊙.test.io
    ⎕ '⊙.test.io'
    ⊙.test.io0 1000
    ⊙.test.io1 0
// ⊙.test.io 0              // nb. test is NIU to not delay startup
`

r += '\n// Document Object Model (dom) support'
r += ⌽.er `
▽ mcode.dom : w             // a few common operations on the HTML DOM

    ▽.a rd : g              // async fn to read html and insert into element
        g = ⍃ ω ; α.innerHTML = g ; ⍔ α

    // document ops, ω is data
    α ⇋ θ →
        δ ⇋ θ           → ⍔ document.getElementById(ω)
        δ ⇋ 'body'      → ⍔ document.body.insertAdjacentHTML('beforeend',ω)
        δ ⇋ 'head'      → ⍔ document.head.insertAdjacentHTML('beforeend',ω)
        δ ⇋ 'css'       → ⍔ document.head.insertAdjacentHTML('beforeend','<link href="'+ω+'" rel="stylesheet" />')
        δ ⇋ 'js'        → ⍔ document.head.insertAdjacentHTML('beforeend','<script src="'+ω+'" ></script>')
        δ ⇋ 'el'        → ⍔ document.createElement(ω)
        δ ⇋ '+'         → ⍔ document.body.append(ω)
        δ ⇋ 'qs'        → ⍔ document.querySelector(ω)
        δ ⇋ 'ael'       → ⍔ document.addEventListener(...ω)     // ω = [ type, listener, useCapture ]
        δ ⇋ 'docwr'     →
            w = window.open('')
            w.document.write(⍵)
            w.document.close()
            ⍔ 0

    ⋄
        // α is id or HTML node, ⍵ is data
        α ⍷ '' →
            α = document.getElementById(α)                       // convert α id to node
            !α → ⍔ 0

        δ ⥊ θ →
            α.innerHTML = ⍵ ; ⍔ α

        δ ⥊ 'el'        →                                        // new el ⍵ with class α, rtns el
            w = document.createElement(⍵) ; w.classList.add(α); ⍔ w

        δ ⥊ '+'         → ⍔ α.appendChild(⍵)
        // δ ⥊ 'rm'        → ⍔ α.removeChild(⍵)
        δ ⥊ 'rm'        → ⍔ α.remove()

        δ ⥊ 'cl?'       → ⍔ α.classList.contains(⍵)
        δ ⥊ 'cl'        → ⍔ α.classList.add(⍵)

        δ ⥊ 'ael'       → ⍔ α.addEventListener(...⍵)             // ⍵ = [ type, listener, useCapture ]

        δ ⥊ 'rd'        → ⍔ α rd ⍵                               // read html from file ⍵, append to el α

        δ ⥊ 'attr?'     → ⍔ α.getAttribute(⍵)
        δ ⥊ 'attr'      → ⍔ α.setAttribute(...⍵)                 // ⍵ = [ 'attr' value ]

        δ ⥊ 'dsp'       → ⍔ α.style.display = ⍵
        δ ⥊ 'clr'       → ⍔ α.style.color = ⍵
        δ ⥊ 'bg'        → ⍔ α.style.backgroundColor = ⍵
        δ ⥊ 'brd'       → ⍔ α.style.borderColor = ⍵
        δ ⥊ 'top'       → ⍔ α.style.top = ⍵
        δ ⥊ 'left'      → ⍔ α.style.left = ⍵
        δ ⥊ 'w'         → ⍔ α.style.width = ⍵
        δ ⥊ 'h'         → ⍔ α.style.height = ⍵
    ⍔ 0
'⍛' ⍙ 'mcode.dom'
`

r += `
// end core

`
```

```
mcode.cp = {}           // clear context
mcode.cp.core = r

⍔ '  core loaded, tests passed'
```