Macaulay2 Language

Arithmetic `+`, `-`, `*`, `^` do what you expect

`/`	division
`//`	integer division
`%`	modular remainder (≥ 0)
`sqrt`	square root
`!`	factorial

Packages

- `loadPackage` (re)loads a package
- `needsPackage` loads a package if not already loaded
- `loadedPackages` lists loaded packages

Lists & Sequences

`List`	`Sequence`
`{1,2,"hi"}`	`(1,2,"hi")`
vector operations	no vector ops
`flatten`	`splice`

Both are immutable & zero-indexed. If `L` is a `List` or a `Sequence`, can...

- Use `#` to access items (via `L#0`) or get length (via `#L`)
- Use `_` to get (multiple) items, via `L_0` or `L_{0,1,4}`
- `append(L,"last")`, `prepend("first", L)`, `insert(2, "middle", L)`
- `drop(L,{2,2})` removes item at *index* 2; `delete(L,2)` removes all items with *value* 2

If `f` is a function, the following are variants on the idea of looping through & applying f

- `scan(L,f)` applies f to each element & discards return
- `apply(L,f)` returns list of f applied to each element

When `f` takes 2 args...

- `fold(f,L)` iteratively applies f to next elem & prev result. `fold(L,f)` does same but starts at end of list.
- `accumulate(f,L)` is like fold but returns list of intermediate results. `accumulate(L,f)` does same but starts at end of list.

If `g` is a `true/false` function, can get sublist/sequences: `select(L,g)`, `positions(L,g)`, and to count, `number(L,g)`

Defining Functions

Example syntax	What it does
`f = x -> x^2`	$f(x) = x^2$
`g = y -> (i:=2; i*y)`	$g(y) = 2y$
`a = (r,s) -> r+s`	$a(r,s) = r + s$

- `;` separates statements
- Need () around body if multiple statements
- use `:=` instead of `=` inside (unless you *want* global variable)

Control Structures For `B` boolean expression, `m`,`n` integers...

- `if B then X else Y`
  Eval `B`, if true eval & return `X` else do same for `Y`. If omit `else`, that branch evals to `null`
- `while B list X do Y`
  Eval `B`, when true, eval `X` and save; eval `Y` and discard; repeat. At end, return list of `X` vals. Can omit `list X` or `do Y`
  Ex: `i=0; while i<3 list i^2 do i=i+1` returns `{0,1,4}`
- `for i from m to n when B list X do Y`
  Inits `i=m`, and as long as `i` $\leq$ `n`, continue. Eval `B`, if true continue. Eval `X` and save; eval `Y` and discard; repeat. At end, return list of `X` vals. Can omit `when p` and/or `list X` and/or `do Y`

Getting help

Command	Usage & Comments
`help`	use alone to get generic menu; use with function to get documentation for that function; combine with below commands
`methods`	`methods X` for `X` function, type, keyword, or package lists methods "associated" with `X`; `methods(X,Y)` for types `X` & `Y` gives methods involving both
`code`	use with (list of) functions to display code, or combine w/ `methods`
`about`	get (list of) relevant documentation bits for string, function, symbol, or type. Combine w/ `help`
`apropos`	get list of global symbols matching string. Allows regex; case sensitive.

Math in M2

Rings & Ideals

Built-in "coefficient rings" are:

- Exact: `ZZ`, `QQ`, `ZZ/p`, `GF(p^n)` (for p prime)
- Inexect: `RR`, `CC` (use `ii` for i)

For ring `R`, define 4-variable polynomial rings via:

- `R[alpha,beta,gamma,delta]`
- `R[w..z]` or `R[vars(22..25)]`
- `R[4]` (gives subscripted vars)

[[vars must be SYMBOLS, use **symbol x** if you've say already defined x to be something]]

Can use "options" (`OptionName => OptionValue`) to alter things. E.g., `ZZ[x,y, MonomialOrder => Lex]`

To make ideal, `ideal`. Ideal operations:

`+`, `*`, `^`	add, multiply, powers
`isSubset`	containment
`==`, `!=`	check equality
`:`	colon ideal

For f ring element, I ideal:

`f % I`	reduce ("remainder mod I")
`f // gens I`	decompose into combo of gens

Other rings/fields:

- Quotient rings: `R/I` for `I` ideal, or `R/s` for `s` sequence of ring elements
- Fraction field: `frac R` for R domain
- If `R` is a field but M2 doesn't realize, use `toField R`
- Tensor products: `R ** S` or `tensor(R,S)`
- Exterior algebra: make poly ring w/ option `SkewCommutative => true`
- Symmetric algebra: for `M` module, `symmetricAlgebra M`
- And more! Weyl algebras, associative algebras, and local rings (see package `LocalRings`)

Working with multiple rings

- `use R` makes `R` current ring
- for `R` a "basering" of `S` and $f \in S$, `lift(f,R)` views f as element of R, if possible
- for `R` a "basering" of `S` and $g \in R$, `promote(g,S)` views g as element of S

Miscellaneous

- Use `substitute` or `sub` to (partially) evaluate polynomials.
  Ex: `sub(x*y+z, {x=>2,z=>3})` gives $2y + 3$
- Use `gens` and `vars` to access generators, as list and matrix (respectively)

Maps & Matrices Use target before source!

- `map(S,R)` for rings gives "identity" map $R \to S$, i.e., tries to match up variable names & sends to zero if can't
  Ex: `R = ZZ[w,x,y]`, `S = ZZ[x,y,z]` then `map(S,R)` is $w \mapsto 0$, $x \mapsto x$, $y \mapsto y$
- For `d` list of images (or list of options) `map(S,R,d)` is map defined by `d`
- Similar for other kinds of objects (modules, chain complexes)
- Matrices: use double-nested list & `matrix`
  Ex: `matrix {{1,2},{3,4}}`

Modules

- To view ring or ideal as module, use `module`.
- Given matrix, use `ker`, `coker`, and `image`
- Create submodules & quotients using expected math notation

The `TestIdeals` package

- `frobenius(e, I)` or just `frobenius(I)` for $e = 1$
- `isFPure(I)` checks if S/I is F-split; `isFPure(R)` checks if R is F-split