Writing Functions — Lab 2

There are many levels at which one can work in *Macaulay 2*. In increasing order of sophistication we have:

1. typing information in to the M2 session.
2. typing information into a file and using F11 to execute commands in *Macaulay 2*.
3. load a file.
4. load or install a package.

In our introductory session we discussed the first two. In this session we will discuss the third. On Friday we will discuss how to put it all together in a package. The type of files we construct today are good for work you don't expect to share with others, or is so early in development you might not want to work with a package. However, for debugging, and any work you might share, a package is by far the best format!

For the following experiment with each in *Macaulay 2*. You can either just type them in or use F11.

1. **Most basic function.**

   ```
   f = (x,y) -> x+y
   ```

   Try f(2,3) now.

2. **Another basic function.** Try the following. Notice the type of assignment we introduce you to for functions with more than one output.

   ```
   f = t -> (t^2, t^3, sqrt t)
   (u,v,w) = f(4)
   v
   ```

3. **A more sophisticated function with some discussion of assignment.** In *Macaulay 2* = is global assignment and := is local assignment. With the caveat, that once a variable is locally assigned the idea is that a local box with that name is made, and if you want to change what is in that box you need to use =, that is we don't want to use := again as that will try to create a second local box with the same name, which is not a good idea.

   ```
   f = t -> (
        R := ring t;
        M := R^3;
        t*M)
   ```

   This function takes an ideal as input and returns the submodule of the free module R^3 given by $t * R^3$. Discuss the relative merits and drawbacks of this function.

4. In *Macaulay 2* we can use if, while and for (conditionals and loops). The page on "The Macaulay 2 Language" is a nice guide. Here are 3 practice problems. Feel free to write a function that seems useful to you from today's lectures rather than these.

   (a) Write a function that prints the minimal primes of a monomial ideal, if the ideal is in fact entered as a monomial ideal.

   (b) Write a loop that successively prints the matrices in a free resolution.

   (c) This function is from the Binomials.m2 package. Figure out what this function does. It uses a while loop, so you see that structure. It also illustrates the assignment discussion with I1, I, and s.

```
axisSaturate = (I,i) -> (
-- By Ignacio Ojeda and Mike Stillman
-- For computing saturations w.r.t. a single variable:
-- Comments by TK:
    R := ring I;
    I1 := ideal(1_R);
    s := 0;
    f := R_i;
    while not(I1 == I) do (
s = s + 1;
I1 = I;
-- This should be just the quotient. Is this faster ??
I = ideal syz gb(matrix{{f}}|gens I,
            SyzygyRows=>1,Syzygies=>true););
    {s-1, I}
    )
```

5. *Macaulay 2* has an additional function structure called a method. Methods allow us to enter much more information about a function and allow the same name for different types of functions. If you make a package (Friday's session), then (at least at the final stages) every exported function *must* be a method. Two other reasons to make a method: optional arguments, and functions that apply to multiple input types, like a module and an ideal.

   (a) For a first example we turn an earlier function into a method. Note that we now identify the output as being of type Module and the input as type Ideal.

   ```
   f = method()
   f Module := Ideal => t -> (
        R := ring t;
        M := R^3;
        t*M)
   ```

   (b) We write that method to work on multiple input types.

   ```
   f = method()
   f Module := RingElement => t -> (
        R := ring t;
        M := R^3;
        t*M)
   ```

   (c) Finally, to allow an optional argument, is not so obvious for this particular method. However, it comes up often. For example when the function constructs a ring using new variables, like reesAlgebra or integralClosure, one wants the user to be able to indicate the name of the new variable, but also to have a default available. Or in the case of binomialCellularDecomposition there are two optional arguments. We won't put the full function here, just the first two lines so you get the idea of the syntax.

   ```
   binomialCellularDecomposition = method (Options => {returnCellVars => false, verbose=>true}
   binomialCellularDecomposition Ideal := Ideal => o -> I -> (
   ```

   Then to call the optional arguments they are o#returnCellvars and o#verbose, for example

   ```
   if o#verbose then (
        << "redundant component" << endl;
        )
   ```

6. Experiment with your own. Think about some process you might want to repeat many times to experiment with to answer exercises, or start on group projects, or just something you've been thinking about. If you need suggestions, ask us.

7. Finally, you might want to write a couple of functions and load them all into *Macaulay 2* at once. Save the functions in a file with the suffix .m2, for example myfunctions.m2. Then in a *Macaulay 2* session type

```
load"myfunctions.m2"
```

Note that to load packages we use loadPackage and leave off the suffix .m2 and to load files we keep the suffix. Also, this is easiest if you started *Macaulay 2* in the same folder. If not, you have to figure out the path and do something like

```
load"/Users/ataylor/everyday/research/colaboration/IrenaSwanson/constructionPrograms.m2"
```

This is just the path to one of my personal files for my work with Irena, but hopefully you get the idea. And, in this file most of the functions need the package Binomials.m2 so I have a line in there that is loadPackage"Binomials".