MACAULAY2 - RTG SEMINAR

OCTOBER 4TH, 2023

We'll try to make a package today.

Step 1 (Make a package).

You can grab a file from my website:

```
https://www.math.utah.edu/~schwede/M2RTG/TempPackage.m2
```

Take a look at that file. Most of the initial contents are self explanatory, or not important for now.

Once you have downloaded this package to somewhere you can access it, start up Macaulay2, and run the command

```
needsPackage "TempPackage"
```

Note, you *do not* include the ".m2" part of the filename. If you rename your file, it *will not work*. The FileName `TempPackage.m2` must coincide with the title of the package (in the `newPackage` command).

There are two main commands to load a package. They are

- `needsPackage:` Will load a package if it is not already loaded.
- `loadPackage:` Will load the package. If it is already loaded will create an error unless the `Reload=>true` flag is set (as it was in `TempPackage`).

If you are writing a package, you need to either `restart` Macaulay2 whenever you change your package, or use the `loadPackage` command.

If you are debugging, just `restart` Macaulay2.

Exercise 1. *Mess around with these functions. Modify* `TempPackage.m2` *so that* `Reload=>false`*. Then try* `loadPackage` *again. Try running* `help TempPackage`*, modify the version number / authors, and try running the* `needsPackage` *and* `loadPackage` *command, see if you can see the version number change.*

Step 2 (Some common mistakes).

Let's learn some common errors. When you load a package it does a number of checks.

Exercise 2. *Try adding the following code to the* `secondFunction` *code, before the conditional.*

```
myVariable = 0
```

This should give an error when you try to load the package. It's telling you where the error is. The way you probably want to fix it is to use the "local assignment" `:=`*. You can also declare* `myVariable` *as a local variable first, by adding*

```
local myVariable;
```

before you set it.

Exercise 3. *Load the package and try executing* `thirdFunction`*. Why can't you execute it? Try fixing it.*

Step 3 (Add functionality)

Exercise 4. *Try adding a new* `Type`*, and a constructor for it. Try using it – figure out what you will need to add to the* `export` *list.*

Exercise 5. *Try adding some* `Option`*s to your method functions. Test them out. You may need to export things as well.*

Exercise 6. *Add the commands* `PackageImports => Divisor,` *and* `PackageExports => Pullback,` *to your package immediately after the line* `Keywords => "A word",` *Figure out what those commands do using Macaulay2's help.*

Step 4 (Documentation)

One of the more important things to do is to write documentation for your users. In the sample file, `firstFunction` is documented.

You can install the documentation (and also check for missing documentation) by running the command

```
installPackage "TempPackage"
```

After installing it, you will want to run

```
uninstallPackage "TempPackage"
```

Installing the package will run all the examples, store the output, and make nicer linked documentation pages. (For example, before and after uninstalling the package run `viewHelp firstFunction`).

Exercise 7 *Write documentation for* `secondFunction` *and the package itself (ie, create a* `TempPackage` *doc node). You can do this by copying the entire* `doc ///` *command. The tabbing (spaces actually) are super important for how the documentation is processed by Macaulay2. Pay close attention to the example. I would recommend spaces instead of tabs in your editor (talk to me if you don't know what I am talking about!).*

Step 5 (Tests)

You can see one test in the file. To run it, execute:

```
check TempPackage
```

Make sure you understand what `assert` does, try `assert(true)` and `assert(false)`.

Exercise 8 *Write tests to make sure all the functions are working as intended. You can put everything in one test (with multiple assert), or write separate test nodes* `TEST ///`*, perhaps one for each function.*