

Self-Reference Module

[Help](#)

Good news! This module's programs are **much** shorter than those from HtDW and Compound.

But it's still an exciting module. We are going to explore one of the most beautiful concepts in Computer Science: well formed self-reference. Of course just the name doesn't mean much. But what it allows us to do is represent arbitrarily large amounts of information. You may not have noticed, but so far we can't do that. We can represent a single city name, but not all the cities on the Weird Sisters Summer 2013 Tour. We can make one cow walk back and forth across the screen, but not a whole group of cows (much less a herd).

Arbitrary-sized information is information that we don't know the size of in advance. A program that can display any number of cows is operating with arbitrary-sized information.





The videos this module range in length. But there isn't a half hour one! As before, the longer ones are mostly just a step-by-step working through of a problem, so while it takes time to watch, it isn't all new content.






























The material in this module should take **approximately 4-6 hours** of dedicated time to complete, including working along with the lecture videos, doing the practice problems, doing the homework problems and the short quiz.

Learning Goals



- Be able to use list mechanisms to construct and destruct lists.
- Be able to identify problem domain information of arbitrary size that should be represented using lists and lists of structures.
- Be able to use the HtDD, HtDF and Data Driven Templates recipes with such data.
- Be able to explain what makes a self-referential data definition well formed and identify whether a particular self-referential data definition is well-formed.
- Be able to design functions that consume and produce lists and lists of structures.
- Be able to predict and identify the correspondence between self-references in a data definition and natural recursions in functions that operate on the data.



Lecture Videos, Notes and Starter Files

Topic	Length (mm:ss)	Starter File	Downloads
Self-Ref - Introduction to Arbitrary Sized Data This week we focus on how programs can represent arbitrary amounts of information. This includes things like all the students in a course, all your friends on Facebook, or (sadly) all your favorite hockey teams.	1:22	<i>none</i>	   




Self-Ref - List Mechanisms Primitive data and operations for representing lists.	8:40	<i>none</i>	   
Self-Ref - List Data Definition A data definition for representing lists of Quidditch teams. In this first list data definition we make a few lucky (and unexplained decisions) and wind up with a data definition that seems to do the job. The mysteries are all resolved in 5e.	11:33	quidditch-starter.rkt	    
Self-Ref - Function Operating on List A function for operating on a list. In this first function that operates on a list we use the data definition from 5c and our function works out splendidly. The mysteries are all resolved in 5e.	7:36	<i>none</i>	    
Self-Ref - Revising the Recipes for Lists The luck of the last two videos is explained. The key ideas of well-formed self-referential data definitions, natural recursion and trusting the natural recursion are the key to operating on arbitrary-sized data. We update the recipes to include them.	12:15	quidditch-recap-starter.rkt	    
Self-Ref - Designing with Lists Examples of using the revised recipes beginning to end.	13:04	designing-with-lists-1-starter.rkt	    
Self-Ref - Positions in List Templates Looking over all the list functions designed so far we see that each position in the list template plays a specific role.	7:15	<i>none</i>	    

Practice Problems

Module Kind #	Assignment	Duration	Difficulty	Code Files	Requires Lecture
Self-Ref P1	Design a function to add '!' to each string in a list of strings.	18 min.		yell-all-starter.rkt yell-all-solution.rkt	Self-Ref - list-fun
Self-Ref	Design a function to find the largest	18 min		largest-starter.rkt	Self-Ref -

P2	number in a list of numbers.	10 min.		largest-solution.rkt	des-w-list
Self-Ref P3	Design a data definition to represent a list of images, and a function to find the sum of areas from a list of images.	35 min.		image-list-starter.rkt image-list-solution.rkt	Self-Ref - des-w-list

Homework Problems

Module Kind #	Assignment	Duration	Difficulty	Code Files	Requires Lecture
Self-Ref H1	Design a function to calculate the total number of individual characters in a list of strings.	18 min.		total-string-length-starter.rkt	Self-Ref - list-fun
Self-Ref H2	Design a function to double every number in a list.	18 min.		double-all-starter.rkt	Self-Ref - des-w-list
Self-Ref H3	Design a data definition to represent a list of booleans, and a function to determine if all values in a given list are true.	35 min.		boolean-list-starter.rkt	Self-Ref - des-w-list

Lecture Problems

Module Kind #	Assignment	Duration	Difficulty	Code Files	Requires Lecture
---------------	------------	----------	------------	------------	------------------

Module Quiz

Be sure to complete the homework problems before you do the module quiz. The quiz itself can be found on the [All Quizzes](#) page.

Tips for Success

The programs are shorter in this module, but it's still important to practice a lot. Be sure to design at least a few of each kind of function: consuming a list, producing a list, consuming a list of structures.

Trust the method to guide you through these problems.

Always trust the natural recursion.