[1] Now we're going to look at 4 of the simplest data structures. They are the bunch, the set, the string, and the list. There's just 2 data structuring principles here, namely packaging, and indexing. The bunch is the simplest structure there is. It's [2] unpackaged and unindexed. The set is [3] packaged but not indexed. The string is [4] indexed but not packaged. And the list is [5] both packaged and indexed. We'll start with [6] bunches. A bunch is a mathematical data structure that can be used to represent a collection of things. [7] And here is a bunch of 3 numbers. And just to show that it doesn't have to be all the same kind of thing, [8] here is a bunch of 2 binary values, a number, and a character. And [9] here is a bunch consisting of only 1 number. So I guess the obvious question here is: how can you tell if you're looking at a bunch consisting of 1 number, or at the number itself? And the answer is: there's no difference between a bunch of one thing, and the thing. That's one of the ways that bunches are different from sets. A set containing the number 2 is not the same as the number 2. But the bunch consisting of the number 2 *is* the same as the number 2. A bunch is so simple that it hardly needs a name. I could just say [10] there are 3 numbers, and [11] there are 4 - whatever, and [12] there is a number. [13] Any number, or binary value, or character, or set which we'll get to later, is an elementary bunch. Or, an element. That means the same thing as elementary bunch. Or a 1 element bunch. [14] If you have 2 bunches and you put a comma between them, you have a bunch consisting of the elements of both bunches. The comma is the bunch union operator. [15] An upside down comma is the bunch intersection operator. [16] The cent sign is the size or cardinality operator for bunches, telling how many elements form the bunch. [17] And colon is the bunch inclusion operator. *A* colon B is true if all the elements of A are also elements of B. [18] I've been trying hard here not to say "elements contained in a bunch". Elements are contained in a set. A set is a kind of container, or package. But a bunch isn't a container or package. The bunch really is the elements themselves.

[19] This equation is just an example to say that it doesn't matter what order you write the elements in, and it doesn't matter whether you write them once each or more than once. [20] The size of an elementary bunch is 1, and the size of a non elementary bunch isn't. The parentheses are there for the same reason we always use parentheses - to indicate precedence. On the precedence table, size comes before comma, so we need the parentheses. [21] And it wouldn't matter if we write elements more than once; that doesn't change the size of the bunch. [22] This says that 2 is included among 0 2 5 9. [23] This says that 2 is included in 2. And [24] this says that 2 and 9 are included among 0 2 5 9. Those were just examples. [25] Here are some of the axioms that define bunches. In these axioms, small letters are elements, and capital letters are any bunches. [26] The first axiom says that for elements, colon is the same as equals. [27] The next axiom says that an element is included in a union if and only if it is included in at least one part of the union. The [28] next one says comma is idempotent. That's why it doesn't matter how many times you write an element in a bunch. And the [29] next one says comma is symmetric, or some people say commutative. And that's why it doesn't matter what order you write elements in a bunch. And [30] the next one says comma is associative, and that's why I don't bother to write parentheses when I have 3 or more elements, because all ways of putting in parentheses are equivalent. And you can look at these other axioms when you have time; they look a lot like binary laws. [31] [32] Let me just show you the rest of the axioms. There, that's all of them. [33] The first 3 say that colon is reflexive, antisymmetric, and transitive. That means colon is an ordering. Colon is the ordering on bunches, just like less-than-or-equal-to for numbers, and implies for binary values. [34] [35] And then there are lots of laws that can be proven from the axioms. I'll just [36] point out 2 of them that say union and intersection are both monotonic. That means that if you make one part of a union bigger, you make the whole union bigger, or at least not smaller. And the same for intersection. Again, these laws look a lot like binary laws, with union being like disjunction, intersection being like conjunction,

and colon being like implication. Look at them more when you have time. [37] Right now I want to list some [38] bunches that are worth naming. The [39] first one is null, the empty bunch. Then [40] bin, which is the two binary values. [41] Then nat, which is the natural numbers, 0 1 2 and so on. [42] int is the integers, which includes negatives, zero, and positives. Then [43] rat, which is the unfortunately named rational numbers, and [44] real, which is the real numbers. The dots all over this page mean "guess what goes here". They're not part of the formalism, so none of these is a formal definition – except for bin. The formal definitions are all in the textbook. [45] The next 4 bunches, xnat, xint, xrat, and xreal, are extended versions of the previous 4. They just have infinity included. [46] And finally char is the characters.

[47] There's just one more bunch notation: x comma dot dot y, which I pronounce [48] x to y, and I use it only for extended integers x and y – that means integer or infinity, and only when [49] x is less than or equal to y. And it stands for the integers starting at x and continuing up to but *not* including y. [50] Well, that's a rough statement; [51] here's the precise statement. Integer i is included in x to y if x is less than or equal to i, and i is less than y. This is an axiom, and comma dot dot is a formal notation. We're not guessing what's included; we have an axiom telling us exactly what's included. Using only 1 comma like that is supposed to help us remember that the left endpoint is included, and the right endpoint isn't. [52] So the size of this bunch is y minus x. There's no plus 1 or minus 1 to worry about. For example, [53] 0 to 3 consists of 3 elements. [54] 0 to 2 consists of 2 elements. [55] 0 to 1 consists of 1 element. And [56] finally, 0 to 0 consists of 0 elements. And [57] just one more example, 0 to infinity is nat.

[58] One of the things that makes bunches useful is the distribution property. Most operators distribute over bunch union. [59] For example, the negation of a bunch of numbers is the bunch of negated numbers. [60] If you add 2 bunches of numbers, you get a bunch of sums. You get all the numbers that are the sum of a number from the first bunch plus a number from the second bunch. [61] Here's a sum of bunches where one of them is an elementary bunch. And [62] here's an example where both bunches are elementary bunches. Or is it just the sum of 2 numbers? However you want to say it, 1 plus 10 equals 11. Now [63] here's an example where one of the bunches is empty. Any guess about the answer? — If either operand is empty, then the [64] result is empty. The number of elements in the result is at most the product of the number of elements in the 2 operands. — This distribution property makes it easy to express lots of bunches. For example, [65] nat plus 2 is the plural naturals, and [66] nat times 2 is the even naturals, and [67] nat squared is the square naturals, and [68] 2 to the nat is the powers of 2, and so on. Lots of operators distribute over bunch union. You can find which ones by looking at the last paragraph in the textbook. — Well, that's it for bunch theory. I think it's the simplest data structure there is.

[69] Now let's look at set theory. If you have some apples lying loosely on a table, that's a bunch. If you put them in a paper bag, now it's a set of apples. There's no difference between an apple and a bunch consisting of 1 apple. But there is a difference between an apple and a bag with an apple in it. With sets you get the power to make nested structures, which means things within things, and that turns out to be so powerful that it can express all of mathematics. To make a set, [70] you start with a bunch, and apply the set formation operator, which is curly brackets. Just like comma, set brackets are also an operator, not just syntax. And it has an inverse, too. [71] The contents operator. If you apply it to a set, you get back the contents of the set, which is a bunch. [72] Here's a bunch consisting of 3 elements. [73] Now it's a set containing 3 elements. But that set is an elementary bunch, or element. [74] Now I'll add another element, so we have a bunch of 2 elements. [75] And we apply set formation and we have a set containing 2 elements, one of which is a set containing 3 elements. And the whole thing is a 1 element bunch, or element. [76] The empty set ends up looking like this, which isn't the standard notation, but I can't help it. And

the set [77] of naturals looks like this. [78] The set containing the first 3 naturals can be written either of these ways. [79] Here the contents operator gives us the contents of a set. [80] The dollar sign is being used as the set size or cardinality operator, instead of the standard pair of vertical lines. And [81] the power operator gives all sets whose elements come from its operand. Well, those are just examples. [82] Here are the axioms. That's all the axioms of set theory. And in fact, some weird set theoreticians don't use [83] this one. That's called non well founded set theory, which sounds bad, but it's actually interesting. What I find amusing is that [84] these 3 axioms, defining elementhood, subset, and power, are all just bunch inclusion. That's it for sets. We'll do strings and lists next time.