Unit 2
Introduction to Data Structures

Unit Overview

Students discover the need for data structures, and practice defining them, making examples, and writing functions that produce them.

Learning Objectives:

- Students will deepen their understanding of function definitions and the Design Recipe
- Students will understand the limitations of atomic datatypes
- Students will write complex functions that consume, modify and produce structures

Agenda

20 min	Review
10 min	Introducing Structs
25 min	Autos
5 min	Accessor Functions
25 min	Autobody Shop
5 min	Closing

Product Outcomes:

- Students identify real-world behaviors that require data structures
- Students make use of a complex data structure: auto
- Students define variables bound to autos
- Students write code that extracts each field from those autos
- Students define functions that produce an auto

State Standards See our Common Core Standards Table provided as part of the Bootstrap curriculum.
Length: 90 minutes

Materials and Equipment:

- Computers w/DrRacket or WeScheme
- Student workbooks
- Design Recipe Sign
- Language Table
- Structs bags: plastic bags containing eight cards (2 labeled "number", 2 "string", 2 "image", and 2 "boolean")
- The Autos file [Autos.rkt from source-files.zip | WeScheme] preloaded on students' machines

Preparation:

- Language Table Posted
- Seating arrangements: ideally clusters of desks/tables

Review (Time 20 minutes)

- *Welcome back! You guys did a great job on the review in our last session, and I think you're starting to get good at the design recipe again.*
- Let's quickly walk through one example as a class, and then we'll have a competition. Turn to Page 7 in your workbook. *Use a projector, so kids can see the function being written on the computer:*
- Write a function called `double-radius`, which takes in a radius and a color. It produces an outlined circle of whatever color was passed in, whose radius is twice as big as the input.
- Step 1: Contract and Purpose Statement
  - Okay, what's the name of the function? double-radius! We know it takes in a radius and a color, so what is double-radius's domain? A number and a string! And what do you think the range is? Image! And what does it do? Can you tell me in English?
  -
```
; double-radius: Number String -> Image
; makes an outlined circle that's twice the radius
```

- Step 2: Examples
  - In your workbook, you will need to write at least two examples of how someone would use this function, and then what those expressions should become.
  - How would someone else use `double-radius`?
  - We can look at the contract we just wrote to figure it out! We know that the name is `double-radius`, so we can call it using that name. If I write `(double-radius 50 "pink")`, describe the circle I should get back.
  - Outline, 100, pink. And how would I make that circle, using Racket code?
  - Can you give another example, and tell me what circle I'd get back?
  - What is changing?? Circle everything that's different between the second part of these two examples: the size! *(write the name, to label it)*
  -
```
(EXAMPLE (double-radius 50 "pink")
(circle (* 50 2) "outline" "pink"))

(EXAMPLE (double-radius 918 "orange")
(circle (* 918 2) "outline" "orange"))
```

- Step 3: Definition
  - How do we start our definition? `define`! Then we look back at our examples for help.
  - What's the name of the function? How many variables do we have? How do you know? It's in the contract!
  - The variables are circled, because they change from example to example; now we need to name them. What did this variable represent? Right, the size! We already labeled it.
  - Now we can just copy one of our examples, and replace the changing thing with our variable! We didn't really have to do any work at all!
  -
```
(define (double-radius radius color)
(circle (* radius 2) "solid" color))
```

- Okay, are you ready to try one on your own? Turn to Page 8 in your workbooks.
- *I'm going to read the problem statement, and as soon as I'm done I'm going to start the timer. You'll have 5 minutes to finish this. Do not skip a step!! Once you're done, ask one of us to check your work. If they give you the OK, you can turn your monitors on and type in the code. Any questions? Here's the problem:*
- Write a function called `double-width`, which takes in a height and a color. The function produces a solid rectangle, which is whatever height and color were passed in. And its width? Its width is twice the height.
- *Review the solution as a class.*

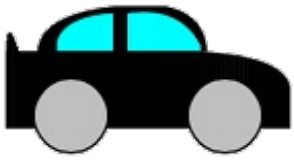
Introducing Structs (Time 10 minutes)

- *Pass out the bags of datatype cards.*
- *Now we're going to play a game...but first I'm going to explain the rules. First off, everyone take out all of the cards from the bags, and set them on the table in front of you.*
- Raise your hand if you can tell me what's written on the cards.
- Right: number, string, image, boolean...but what are these? Datatypes.
- Here's how this is going to work: I'm going to say something that can be returned by a Racket function, and you're

going to hold up a card to show me what datatype it would be. If I were to say age, which would you hold up?
- Now, there's one very important rule: no matter what I say, you can only hold up one thing. Ready?
-
    - a color
    - a picture of a circle
    - your name
    - whether or not something is correct
    - an x-coordinate
    - your friend's favorite food
    - a picture of ninja cat
    - a set of coordinates
- Wait! How many things are in a set of coordinates? An x and a y. Is that one thing? Is it one number? NO! You can only hold one thing up, but we'd need to show two numbers.
- Can we use a String to return two numbers? Not if we want to add or subtract!
- Can we use a Boolean? An Image? None of our data types work!
- It turns out that Racket has exactly this problem. Every function that you could possibly write or use in Racket can only give back one thing. That is, its range only has one thing in it.
- We need a new type - something that can hold more than one thing at once. Racket actually has a tool to make such a thing, and it's called a data structure, or "struct" for short.
- *Set aside the two number cards; one for the x and one for the y coordinates. Then pick up your plastic bags.*
- *Put the two number cards inside the plastic bag, and then hold it up. How many things are you holding? One!*
- In the same way, complex structs can be defined, in Racket, to hold multiple things. Let's keep going with the game, but keep in mind that you may need to use your "struct bag" for some of these. What if your function was returning:
-
    - the name and the age of a character
    - a flavor of soup, and whether it is hot or not
    - how many pets you have
    - a picture of a shape, with the number of sides and its color
    - a direction that a plane is traveling, and how fast it is going
- *Good job guys! Put all of your cards back in the bag and leave it on the table.*

Autos (Time 25 minutes)

- Suppose you want to open up an autobody shop. You take people's cars and trick them out, giving them paint jobs, turbo-charging them, etc.
- What type of thing is an auto? Is it a number? String? Image? Boolean? Of course not. You could not describe all of the important things about an automobile with any one of those things. However, we could say that we care about a couple of things in our shop that can be described with these types.
  Start a list on the board, with the variable name on one side, and the type on the other
- First...the model of the car. That could be like..."Prius", "H2", "XTerra", or something else. What type would that be?
- How about how much horsepower my car has? How large the rims are? What color it is? The value, that the car is worth?
- Write on the board:
    - model: String
    - horsepower: Number
    - rims: Number
    - color: String
    - value: Number

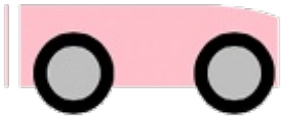


- These are the only things that we're going to keep track of in an auto, but you can imagine how we would extend it to include other things.
- Now that you have a bit more information about them, what data type could we use for this auto? A struct! This is the very first time that we're going to use structs...but they're going to play a HUGE part in your game. Let's take a look at how this works.
- *Have students open the Autobody Shop file.*
- *Raise your hand if you can read the line beginning with* `define car1`. *Have one student read the line aloud, while you write the definition on the board.*
- `(define car1 (make-auto "M5" 480 28 "black" 50000))`
- As you can see here, it's really easy to make this auto struct! We have a bit of code which tells the computer which order everything goes in...and we'll talk about that next class. For now, I want you to look at this new function: `make-auto`. Let's flip to our contracts sheet and write it down.
- What is the name of the function? `make-auto`. How about the domain? How many things are in the domain? Five. They are, in fact, the five things that we have already listed on the board. So what is in the domain of make-auto? String, Number, Number, String, Number.
- *Unlike our struct bags, here the order is important, so be sure to write it down correctly. In fact, why don't you use*

another line to denote what each of these means...the first string is model, the next number is hp, etc. Give them a moment to write down something that looks like this:

- ```
  ; make-auto : String Number Number String Number -> ____________________
  ; Model, hp, rims, color, value
  ```

- Okay, so we have the name and the domain. What's the last part of the contract?
- So what is the range? It's a struct, but what type of struct is it? Take a look at the name of the function, and have a guess. An auto.
- Later we're going to talk about how to make your own structs...Racket obviously doesn't have autos built into it, so later we'll see what part of this code makes it work.
- When I type `car1` into the interactions window and hit enter, what do you think I'll get back? Click run, and then try it out.
- Does this make sense? What happened when we typed a number into the interactions window? We got that same number back! What about strings? Images? Booleans? Right! If we don't do anything to our input...if we don't use any function on it...we get back exactly what we put in! We put in an auto, and we got back that auto.
- Let's define another car; we can call it `new-car`. To start, how will I define this variable?
- `(define new-car ( .....))`
- Now what function do I use to make an auto? `make-auto`.
- `(define new-car (make-auto ...))`
- Which thing comes first in making my auto? Check your contracts sheet if you don't remember: you wrote this down for a reason!
  Have them walk you through the definition of `new-car`...for example:



- `(define new-car (make-auto "Taurus" 300 20 "white" 5000))`
- Now when we type `new-car` into the interactions window, what do you expect to happen? Hit run and try it out. Take a minute with your partner, and define two new variables–one for each of your favorite cars. Call them [yourname]-car (`nathan-car`, `sam-car`, `jill-car`, etc). It can be any kind of car that you want, as long as your struct has the right types in the right orders!
- You can see what your cars look like by using the function we've provided for you at the bottom of the screen. It's called `draw-auto`, and it takes an auto as input and gives you back an Image with your car in it.
  Give them a few minutes to define and draw cars.

Accessor Functions (Time 5 minutes)

- Suppose you want to get the model OUT of `new-car`. The computer has a function for that, called `auto-model`. I can type `(auto-model new-car)` and get out `"Taurus"`.
  Show this on the board.
- I want you to practice taking the model out of your autos. Take a minute and pull the model out of EVERY auto you have, using `auto-model`
- Flip back to your contract sheets. What's the first part of a contract? The name! So what's the name of this function? `auto-model`
- What's the second part? The Domain! What do you think the domain is for auto-model? It's an auto!
- And finally, what's the third part? The Range! So what's the range of auto-model? A string!
- Write the contract for `auto-model` on your contract sheet. What do you think the contract for `auto-hp` is? Write it down too!
- Take two minutes and write down the contracts for `auto-rims`, `auto-color` and `auto-value`. When you show me that you've written them, you can try them out on your autos.

Autobody Shop (Time 25 minutes)

- Now we know all about how to work with automobiles in Racket. What function makes an auto? `make-auto`. And which function draws that auto? `draw-auto`. But we don't just want to take an auto and give it right back. I said that we're running an autobody shop! You take people's cars and change them, making them better in some way, and then give them back to the customer. Let's figure out how to do that.
- Turn to Page 9 in your workbooks. I know that you're really good at the design recipe by now, so I want you to fill out the page as we talk.
  Make sure they're able to keep up, and take time to let them write if need be!
- Let's write a function called `paint-job`, which changes the color of an automobile.
- That's our problem statement.
  ```

- What's the first step in the Design Recipe? Contract.
- What is the name of the function? What does it take in? Remember that we're getting instructions about which color to make it, as well as which auto we're changing the color of! And what do you think our autobody shop is going to give back? What would be the range of `paint-job`?
  Write the contract on your page.
  What's the purpose statement?
- What's the next step in the Design Recipe? Examples!
- Let's use the original `car1` as an example, and let's turn it purple.
- What do we write next? If you're stuck, look up at the range of the function
  : raise your hand if you can tell me what I have to give back
  .
  That's right!
  An auto! And how do I make an auto?
- The moment I write `make-auto`, I immediately know that I have to give it five things: the model, hp, rims, color, and value of that auto. Does the model of the auto change, just because I painted it a different color? No! So how do I get the model out of `car1`?
- The horsepower shouldn't change either with a paint job. So how do I get the hp out of car1?
- The rim size shouldn't change with a paint job. So how do I get the rims out of car1?
- What about the color? Does that change in this example? Yes! It should be the string "purple".
- How about the value? Did our problem statement say that the value changes? So how do I get the value out of car1?
- *Make sure you're copying this down into your notebooks! Give them time to catch up writing.*
- Let's do another example! This time we want to paint car2 green. What do I write for the first part of the example?
- See if you can finish off the rest of this example on your own.
- What's the next step in the Design Recipe? Definition.
  Since you've done such a good job in your examples, this step becomes easy!
  What two things change between these examples? The color and the auto!
  Circle them.
  So how many variables will your function need?
- Write the definition, using the examples to help you. Remember: All you have to do is copy the examples, changing the values to variables!
- *Okay - are you ready to try one on your own?*
- Turn to Page 10 in your workbooks. Now it's your turn.
- When you turbocharge an engine, you get more power out of it. Your bodyshop does a turbocharging job that adds 20 horsepower to any engine, but keeps everything else the same. Write the function `turbocharge`.
  You have 10 minutes. GO!
- *Walk around the class, helping kids as-needed. Don't forget to count down the minutes! If they finish writing, let them put it into the computer.*

Closing (Time 5 minutes)

- *Who can tell us one thing we learned today?*
- *Who saw someone else in the class do something great?*
- *Cleanup, dismissal*