Unit 3
The Autobody Shop

Unit Overview

Students, having made pre-built data structures in the last lesson (autos), are introduced to the syntax for accessing the fields of those structures. They are then forced to generalize the understanding, by defining various data structures of their own and accessing their fields. Students are introduced to Racket's purely-functional microworld implementation. This requires an understanding of functions, data structures, and an introduction to events-based programming. To accomplish this, students first work with a simple world (a number, representing a dog's x-coordinate). This world is consumed and produced by the update-world function, and drawn by draw-world. To understand events, they act out the World model, actually becoming event handlers and timers, to simulate a running program.

Agenda

20 min	Review
5 min	define-struct
30 min	The Party Struct
30 min	Acting Out Ninja World
5 min	Closing

Learning Objectives:

- Generalize their understanding of function constructors and accessors
- Write complex functions that consume, modify and produce structures
- Deepen their understanding of structures, constructors and accessors by being introduced to two new data structures.
- Discover the event-based microworld implementation of Racket, which uses events to modify the world.

Product Outcomes:

- Students define two new complex data structure: party and world
- Students will write functions that access fields of an auto, party, or world, and produce new autos, parties, and worlds.

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*
- *The Autos file [Autos.rkt from source-files.zip | WeScheme] preloaded on students' machines*
- *The Party Planner file [Party.rkt from source-files.zip | WeScheme] preloaded on students' machines*
- *The Ninja World 1 file [NW1.rkt from source-files.zip | WeScheme] preloaded on students' machines*
- *update-world, big-bang, and draw-world nametags*
- *cutout image of dog*

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

Review (Time 20 minutes)

- *Welcome back!*
- Last time we learned about a new kind of data struct, called an auto. Can you tell me what an auto is? An Auto has a model, a horsepower, a rim size, a color and a value. Do you remember how to make an auto? How do I get the model out of an auto? The value? The color?
  Take notes on the board.
- *You used all of these to make an autobody shop, where you had functions that would increase the auto's hp, or paint it a new color. This next problem's going to be even harder...so remember that the last two you wrote are written in your workbook, in case you need some hints.*
- Do you know the show "pimp my ride"? Let's implement that in Racket. Suppose we want to pimp out a car - we want to make it WAY cooler!
- Turn to Page 12 in your workbooks. Pencils in hand! When everyone is silent, I'll give you the problem statement.
- Write a function called "pimp", which takes in an Auto and gives you a new Auto which has an extra 100 horsepower, has 30 inch rims, is painted red, and has increased in value by $10,000. If you run into trouble and need help with one step of the Design Recipe, you MUST show us that you have finished all of the parts before it. You will have 5 minutes. When you show us that you've finished, you can put it into the computer. GO!
- *Go over the answer with the class*

define-struct (Time 5 minutes)

- *Have students open their Autobody file, and look at the top line.*
- *Raise your hand if you can read the line beginning with* `;an auto is...`. *Have one student read the line aloud. Raise your hand if you can read the line beginning with* `define-struct`.
- I told you last time that we were skipping over the part of the code that defines the auto struct, or tells the computer what an auto is and what goes into it. Just like we would expect from having worked with autos, the `define-struct` line says that an auto has five things....a model, hp, rim, color, and value.
- How do we know which number is which? Why can't the horsepower number be the size of the rims? Because order matters! Look at the order of the fields in our `define-struct` line. The first number is the horsepower, the second is the rim size, and so on.
- When we define a struct using define-struct, we tell the computer what order and type each thing is. In return, we get new functions to use. Until we write this `define-struct` line, we didn't have `make-auto`, to make an auto, `auto-model`, to get the model out of the auto, `auto-hp`, or any of the other accessor functions, because Racket doesn't know what an Auto is - we haven't defined it!
- To check this, I want you to type a semi-colon before the line which begins with `define-struct`. This comments it out, so that the computer ignores it. Hit run, and see what happens.
- You get an error! Racket says that you're trying to use an identifier before its definition. That means that it doesn't know what `make-auto` is, since we took that `define-struct` line out!
- Turn to Page 13 in your workbook, and copy down the define-struct line.

The Party Struct (Time 30 minutes)

- Now that we know how to define our own structs, let's define another one. Instead of working in an autobody shop, this time we're going to be party planners. We need to be able to use data structs to represent each party that we're planning, keeping track of their location, theme, and number of guests.
  As with the five parts of auto, write location, theme and guests onto the board, with space for their respective types.
- What datatype could we use to represent the location of the party? How about a string?
- The name of the theme? Like `"50s"`? Or `"laser tag"`? String. How about the guests? Number.
- So how would we write the next line on page 13? `;a party is a ______`
- Now let's fill out the next bit. Look above, at how you did it for auto. What were the three parts of a party? location, theme and guests. Take twenty seconds to finish that page.
- *When I say go, turn on your monitors and open the party planner file.*
  At the top of the definitions window, take a look at the first two lines. Do they match what you have written?
- Take a minute and define your own party struct. No matter what it is, make sure that it has the right types of things in the right order.
- *5, 4, 3, 2, 1, monitors off.*
  Let's write some more contracts: turn to your contracts sheet. Which function did we use to make a party? `make-party`. What would its contract be?
- Which function did we use to get the model out of an auto? How did we get the color out of an auto? So how would you get the location out of the party? `party-location`.

- Which other two functions can we write, now that we've defined our party struct? Can you write their contracts?
- Now let's write some functions using our party struct. Remember, we're party planners, so we need to be able to change information for each party. Turn to Page 14. Let's do the first one together. Here's your problem statement: Write a function called RSVP, which takes in a party and adds one to its number of guests.
- Alright, so what's the name of the function? Domain? Range? Write the contract and purpose statement on your page.
- What's the next step in the Design Recipe? Examples. Let's write one for the party "Halloween". How would we start our example?

-
  ```
  (EXAMPLE (RSVP "Halloween")....)
  ```
- Okay, now we're stuck. Think for a moment about what we need to get back...what do we need to make? A party.
- Which function do we use to make a party?

-
  ```
  (EXAMPLE (RSVP Halloween)
  (make-party .....))
  ```
- What do we want to happen to the location? Does the location change at all? No, of course not. So how do we get the location out of the party?

-
  ```
  (EXAMPLE (RSVP Halloween)
  (make-party (party-location Halloween)
           .....))
  ```
- What about the theme? If someone new RSVPs, do they suddenly start have to make it a christmas party? No. What should we write?

-
  ```
  (EXAMPLE (RSVP Halloween)
  (make-party (party-location Halloween)
              (party-theme Halloween)
           .....))
  ```
- Lastly, what did we say happens to their guests, when we RSVP? Their guest list goes up by one.

-
  ```
  (EXAMPLE (RSVP Halloween)
  (make-party (party-location Halloween)
              (party-theme Halloween)
              (+ (party-guests Halloween) 1)))
  ```
- Now, on Page 15, here's your next problem: Write a function called relocate, which takes in a party and the location that it's moving to, and makes a new party at that location. Go through each part of the design recipe: contract, examples, definition.

Acting Out Ninja World (Time 30 minutes)

- Do you remember the Ninja Cat game from Bootstrap 1? In this course, we're going to completely deconstruct the game, and re-make it using a world structure to make it more complex.
- This version of Ninja Cat might look a bit different than you remember...
  Open the NW1 file and press "Run" and watch the dog fly across the screen
- What do we see in this world? A Dog. Let's be a little more specific, and look at our world struct. Who can raise their hand and tell me how we've defined our world?
- Our world is just one thing: `dogX`. What does `dogX` represent in the game? What kind of thing is that?
- Take a look at the section labelled STARTING WORLD. We have defined here a variable, START. What is START? It's a world! How did we make this world?
  Write `(make-world 0)` on the board
- Let's skip a bit farther down to where it says "Updating Functions". What is the name of the function defined here? What's it's domain and range?
- Can someone raise their hand and tell me what `update-world` is doing?
- Every time `update-world` runs, it makes a new world, adding 10 to the `dogX` of the original world.
- Now skip down to the last function defined in our code: `big-bang`.
- Big-bang is a special function that will begin an animation, but it needs help from other functions to update and draw

the world.

- Take a look back at our START world, `(make-world 0)`. If I were to evaluate just `(big-bang START)`, what do you think would happen?
Take a few guesses, then type it into the interactions window.
- Nothing happens! We see our background, (make-world 0), but nothing's changing or happening! Let's fix that.
- In our code, `big-bang` is calling on a few different functions. Who can raise their hand and read me the second line in our big-bang function?
- The function `on-tick` acts kind of like a timer, and on each "tick", it updates the world. What did we say was in our world structure? Just one number, representing the x-coordinate of the dog.
- So on every tick, we want `update-world` to update the world. How does it do that? It adds 10 to the `dogX` of the world!
- Try typing `(big-bang START (on-tick update-world))` in to the interactions window and see what happens.
- We've got our world structure updating, and now we need to know how to draw it. Scroll up to where you see "GRAPHICS FUNCTIONS". What is the name of this function? `draw-world`. This function takes in a world and draws us an Image. What is the Domain of this function? The Range?
- Now let's look at the body of the function. It's using a new function you have never seen before, called `put-image`. Raise your hand if you think you know, just by looking at this function, what it does. Take a few guesses.
- `put-image` takes in the first image, and then places it on top of another image, using the x- and y-coordinates to determine where to put it. In this example it is placing the `DOG` onto the `BACKGROUND`. What is it using for the dog's x-coordinate? The dog's y-coordinate?
- OK, let's put all these functions together.
- I need a volunteer to be the `update-world` function.
Give them the "update-world" sign.
What's your name? Your domain? Your range?
- How are you updating the world? Each time you're called on, you'll make a new world, adding 10 to the `dogX` of the original world.
- So if I were to say " `(update-world (make-world 0))` ", what will you return?"
The student should erase the 0 and write a 10. If they are stuck, refer to the code.
- Great job! Let's practice this a few times: "update-world, this world!"
Point to (make-world 10) on the board. Go through a few iterations, updating the new world each time.
- Thank you, `update-world`! Stay here, because I'll need to use update-world again in just a second.
- I need another volunteer to be the `draw-world` function.
Give them the "draw-world" sign, and the dog cutout.
Your name is now "draw-world". How does this function work? You take in the current world, and put the image of the dog on the screen at its x-coordinate (`dogX`), and 400 on the y axis.
- *Draw a large rectangle on the board, representing the game screen.*
- If I were to say "Draw-world (make-world 0)", what would you do?
The student should place the cutout of the dog on the rectangle at (0, 400).
- What about "draw-world (make-world 400)"?
go through a few iterations.
- OK, I need one last volunteer to be `big-bang`, so we can bring all these functions together.
Give them the "big-bang" sign.
Do you remember what you're doing? You start the animation, and you have a timer. The class will yell "tick!" every five seconds, and you're going to tell update-world to update the world, just like I did. Let's try it out - on every tick, I want you to give the current world to update-world, who will then update it and replace it with the new world. Ready? GO!
- But `big-bang` calls on one more function to complete the animation. What are we missing? `draw-world`!
- So, `big-bang`, once `update-world` updates the world, you're going to call on `draw-world` to draw that updated world.
- Let's try it all out. Every "tick" the class makes, `big-bang` will tell `update-world` to update this world on the board, and after that, will tell `draw-world` to draw that world. Ready?
Go through a few iterations, so the class can see the world structure change, and the dog move across the screen as they tick.
- Excellent! Let's give our volunteers a hand.
Reclaim function signs, and have the students return to their seats.
- Fortunately, Racket has the capability to run all these functions and more in a fraction of the time, to create and draw a smooth, complete game. In the next few classes, we'll be using structs to extend this world into an actual, complex game, and writing functions for Ninja World and your own games to make them playable and unique.

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*