Unit 4
The Design Recipe

Unit Overview

Students are introduced to the Design Recipe and apply it to simple problems.

Learning Objectives:

- Students will become familiar with the steps of the Design Recipe.
- Students will practice Racket syntax and the Circles of Evaluation.

Product Outcomes:

- Students,in pairs, will write functions to solve simple problems by using the Design Recipe

Agenda

15 min	Introduction
25 min	Introducing the Design Recipe
20 min	Practicing the Design Recipe
20 min	Even More Practice (Yard Area)
10 min	Closing

State Standards See our Standards Document provided as part of the Bootstrap curriculum.

Length: 90 minutes

Materials and Equipment:

- *Student workbook folders - in pairs! - with names on covers*
- *Pens/pencils for the students, fresh whiteboard markers for the teachers*
- *Class posters (List of rules, language table, course calendar)*
- *Language Table (See below)*

Preparation:

- *Write agenda on board*
- *Display Class posters, Language Table, Design Recipe*
- *"Rocket" [Rocket.rkt from source-files.zip | WeScheme] preloaded on students' machines*
- *Seating arrangements: ideally clusters of desks/tables*

Types	Functions
Number	`+ - * / sq sqrt expt`
String	`string-append string-length`
Image	`rectangle circle triangle ellipse radial-star scale rotate put-image`

Introduction (Time 15 minutes)

- Review material from previous lessons: Coordinate Planes, Circles of Evaluation, Contracts, Strings, Images, and Functions.

Introducing the Design Recipe (Time 25 minutes)

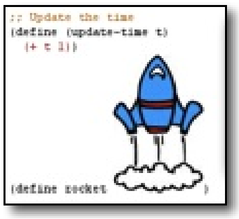
- Now it's time to start thinking about animation, since you'll need it for your videogame!
- You already know most of the steps from the last lesson, but we're going to do a few practice exercises before turning you loose on your own games.
- Ever hear of a flip-book animation?

[(Click here for an example.)]

- When you make a flip-book, you draw each page a little differently from the page before it. We can use functions to do just that! Suppose we had a function that drew a rocket ship a little higher up on each page of a flip-book. What would it look like if you moved through the pages quickly? It would look like the rocket was flying! You've already learned how to draw one frame of your videogame with the characters at certain coordinates, and now we'll learn how a program can change those coordinates. **Today, you're going to learn how to animate a rocket**, then next time you will use what you learned today to animate the characters in your games!
- Believe it or not, you're 90% of the way towards being able to write an animation function! All the steps you took to write those functions before are actually part of a process that real programmers use, called the Design Recipe.
- Engineers all over the world use the Design Recipe to write functions. It's a way of thinking through each step of programming and making sure that you're on the right path before you even touch the keyboard.
- Each step in the Design Recipe tells you a little more about the function you're writing, allowing you to find the answer in pieces instead of having to come up with it all at once.
- We're going to try using the Design Recipe to animate a rocket, and for this you'll be working in pairs.
- First, let's take a look at some BUGGY code that someone else wrote, since they forgot to use the Design Recipe.
- Open the Rocket file, and **click run.**
- A window appears, with our rocket starting out on the ground. At the top of the window, you can see the number of seconds that have passed since blastoff, and the height of the rocket at that time. How many seconds have passed? How high is the rocket?
- Now *hit the space bar to make time go by*. What happens to the rocket? Nothing! This code has a bug in it! Let's see what went wrong...
- Turn to Page 11 in your workbooks.
  Have a student read the Word Problem aloud.
- Just by reading this **Word Problem**, we can tell a lot about the function we're going to write. For example, what is the function's name? Have one person in each pair underline the name on your paper.
- What *kinds of things go into this function?* Numbers? Strings? Images? *What kinds of things does the function give back?* Now the other person in each pair Circle these on your paper.
- *Skit:*

Who would like to volunteer to act out this function?
*What is your **name**?* `rocket-height` *What is your **Domain**?* `Number` *What is your **Range**?* `Number` *When I give you the number of seconds since blast-off, what do you do? multiply it by seven. Let's try it out: "rocket-height three!" (21) "rocket-height zero!" (0) etc.*

- **Step 1: Contract**
- The Contract for a function is like an instruction manual. It tells you the name of function, what types of things it expects, and what type of thing it gives back. It's important to write it down in your code, so that you can always refer back to it when you use that function!
- *Copy these answers onto the board:* `; rocket-height : Number -> Number`
- Copy your contract down into your Design Recipe page!



```
;; Update the time
(define (update-time t)
  (+ t 1))

(define rocket                    )
```

- **Step 2: Give Examples**
- It's always a good idea to work through a few examples, to make sure you understand what the function should do. Then we can use those examples to test our function, to make sure it works the way we expect. You wouldn't build a car without testing it on some sample roads, would you?
- *Skit:*

We already tried some examples with our volunteer.

When we said "rocket-height three"
(write the accompanying code `(EXAMPLE (rocket-height 3) ...)` *on the board)*
, what did we get back? 21! But *how* did we get that? By multiplying 3 and 7. Let's write that code into our example:
`(EXAMPLE (rocket-height 3) (* 3 7))`

- In your notebooks, I want you to come up with another example.
- Now we need to circle what's different between these two examples - what's changeable? What does that number represent? Is it the number of astronauts? No, it's the *number of seconds* since blastoff.
- **Step 3: Code**
- This next part is easy: it's exactly what you did for fast functions! Can you figure out how to plug in the header? What about the body?
- Now let's look at the buggy code. Did they get the contract right? What about their examples? Notice that they only wrote one example, and they forgot to write down how that example worked! If they'd written a second example, they would have immediately seen that something was wrong! But instead, they just raced through the EXAMPLE step and copied the zero into their function body.
- Add the second EXAMPLE you and your partner came up with, and fix the one that's already there. Then fix the code and let's see that rocket fly!
- Once you are done, you and your partner can type this function into your computers.
- *Do you want to see the rocket fly?*
  Type `(start rocket-height)` to see your code in action. We've slowed time down, so that a second passes each time you hit the spacebar.
- *All right. Now let's have a little bonus round.*
- One point to every team who can figure out how to make the rocket fly twice as fast. You'll have 1 minute. GO!
- One point to every team who can figure out how to make the rocket fly backwards. You'll have 1 minute. GO!
- One point to every team who can figure out how to make the rocket fly faster over time, so it gets faster and faster as it gets higher. You'll have 1 minute. GO!

Practicing the Design Recipe (Time 20 minutes)

- *Note: if space allows, this section can also be done on the board with multiple, simultaneous challenges. See the video links for an example.*
- *Skit:*

I need another volunteer, to act out a new function called `red-square`
When I say "red-square fifty", I want you to make a solid, red square where each side is 50 pixels long. "red-square fifty!" "red-square ninety!" etc..

- Turn to Page 12 in your workbooks. You'll have 2 minutes to read the **Word Problem** and to underline the function's name, domain and range. GO!
- *Count down the last 10 seconds, and then collect answers from teams.*
- Now you'll have another minute to fill in the contract for your function. GO!
- *Count down the last 10 seconds then collect answers from teams. Keep score!*
- **Step 1: Contract**
- What's the name of the function we're trying to write? What kind of Domain did we say it expects? Range? Fill these in under as the three parts of your contract. (Answer: "Name: " `red-square` "; Domain: Number; Range: Image")
- *Count down the last 10 seconds, and then collect answers from teams.*
- Now you have to give two example. Again, you'll have one minute. GO!
- **Step 2: Give Examples**
- When given the number 20, what should `red-square` draw? A solid, red square of size 20.
  - How do I write the Racket code for that? (Answer: `(rectangle 20 20 "solid" "red")`)
  - Fill in the boxes with the Racket code for the other examples, including the generalized pattern.
  - Now circle everything that changes between the examples, and label it.
- *Count down the last 10 seconds then collect answers from teams. Don't forget to keep score!*
- Now for the final step! This one is worth two points: one for filling in everything on the "define" line and one for filling in the body of the function. You'll need to fill in the name of your function and a name for the variable. Then you'll need to figure out what goes on the next line for the function's body. You will have two minutes for this. GO!
- **Step 3: Code**
  - As always, we start with the function header: "define", then the name of the function and the names of the inputs from our Domain. What is the name of our function here? What is the name of our input? (Answer: "Name: " `red-square` " ; Input: size (number is the type)")
  - Now you need to fill in the function body. Which pattern can we use for this? The general one.

- *Have kids turn their monitors on, and type in the code. Do their tests work?*

Even More Practice (Yard Area) (Time 20 minutes)

- *If your students seem to be getting the hang of the Design Recipe, you can introduce the WeScheme Design Recipe Widget as described here. If they're still struggling, hold off until they really understand what the Design Recipe is for.*
  Now that you're getting a feel for the Design Recipe, I'd like to show you a way to fill out the Design Recipe *on the computer*, so you don't need to write anything twice.
- In the WeScheme toolbar, click on the button that says "Recipe". A form will appear, with boxes for you to enter the function's Name, Domain and Range. Once you've filled these out, the form will unlock the next step in the Design Recipe: EXAMPLEs!
- Remember that every EXAMPLE has two parts! First, you need to write down an example of that function applied to some inputs. If your friend were acting as one of the functions we've done in the past (green-triangle, blue-circle, etc.), this is what you would say to them to make them do some work. The second part is the code for *the work they would do*.
- Each of these parts much have it's own set of parenthesis, since they each have to be a valid Racket expression. If you make a mistake typing them in, the computer will tell you by outlining the box in red. Click on a red box for hint about what might be wrong.
- When you have two EXAMPLEs written out, the computer will unlock the last part of the Design Recipe: defining the function! Just as before, you can look at your examples and copy everything that *has not changed*, and replace what has changed with a variable.
- When you're all done, you can click the "Insert" button, and WeScheme will take copy your Contract, Examples and Definition right into the editor!
- Doing things on the computer can be a real time-saver, but it's not as flexible as doing them on paper. For the final round, you can choose which method you'd like to use. In the future, you can also choose which one to use – some people find they prefer sticking to paper to think things through, and others prefer working straight on the machine.
- Okay, now it's time for the final round. When I say go, teams will turn to Page 13, and begin. This time, each time will have 5 minutes to Read the **Word Problem**, write the contract, and fill out the examples. One point for each section. Any questions? On your mark ... get set ... GO!
- *Time 5 minutes, count down the last 10 seconds, and then collect answers from teams. Keep score!*
- Now you'll have 5 more minutes to finish the Design Recipe, and to type in your function and test cases. On your mark ... get set ... GO!
- *Time 5 minutes, count down the last 10 seconds, and then collect answers from teams. Keep score!*
- *If time allows, pass out extra Design Recipe worksheets [PDF | Ooo], and run another round: e.g., one of*

```
(define (years-ago year) (- 2012 year))
(define (plural word) (string-append word "s"))
(define (diamond size color) (rotate 45 (rectangle size size "solid" color)))
```

Closing (Time 10 minutes)

- What you have learned is powerful: a structured way to approach **solving any word problem**. First you state the problem as clearly as you can, then think about the data. Then you come up with examples of how the finished product should look, and write a template using only what you know. Only then do you solve!
- Engineers use this when they design cars, by testing each part before putting them together. Journalists use this when they write articles, by making an outline to be sure that each paragraph makes sense before fitting it into a story. Chefs use it when they make food, tasting each part of a dish before assembling it into a meal. Everyone, in every profession, needs to solve problems, and being able to break complex problems down into simpler parts is the key to being good at solving them. To succeed in this class, you need to be masters of solving Word Problems. Fortunately, you've got the Design Recipe on your side. Once you've got the recipe down, you'll be able to tackle anything!
- Next time, you will use the Design Recipe to animate the characters in your game! Don't forget the recipe - it will save your life over the next few lessons!
- *Who can tell us one thing we learned today?*
- *Who saw someone else in the class do something great?*
- Cleanup, dismissal