Unit 3
The Definitions Window

Unit Overview

Students are introduced to the Definitions window, and learn the syntax for defining values of various types. They are also introduced to the syntax of defining functions and creating examples.

Agenda

15min Introduction
10min Defining Variables
30min Screenshots
10min Fast Functions
10min Blue Circle
10min Double
5min Closing

Learning Objectives
Students will:

- Learn about examples, variables and functions
- Practice Racket syntax and the Circle of Evaluation.

Product Outcomes

- Students, in pairs, will write functions to solve simple problems.
- Students, in pairs, will write examples (unit tests) to check those functions.

State Standards
See Bootstrap Standards Matrix provided as part of the Bootstrap curriculum.

Length: 90min

Materials and Equipment

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

Preparation

- ☐ Create student game files. [See Instructions]
- ☐ On student machines: Student Game Files (generated from blank templates [DrRacket file + teachpack | WeScheme]) preloaded, with their image files linked in
- ☐ Write agenda on board, and post along with class posters and the Language Table
- ☐ Seating arrangements: ideally clusters of desks/tables
- ☐ Optional: demo machine with projector to show the interactions and definitions windows
- ☐ Load pre-class assessment on student machines

Pre-Class Assessment: Review of Unit 2

- *This data helps us track student performance. All data collected is anonymous, and participation is optional.* If you would like us to see the assessment results for your class, contact us.

Language Table

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

- You guys have done a fantastic job in the last two classes!
- You've learned how to convert expressions into Circles of Evaluation, and how to convert those circles into Racket code.
- You've learned how to think about functions in terms of nested circles, and how to think of them as a mapping between Domain and Range.
- You've extended that knowledge into three data types: Numbers, Strings, and Images.
- Let's see how much you remember!
- Have students begin by taking the pre-class assessment on their computers.
- You have learned a LOT, and before today's class is over...you will write your first line of code for YOUR videogames!

Defining Variables — Time: 10 minutes

- *NOTE: this section and the one that follows ("Game Screenshots") teach the same concepts as the supplemental Flags and Changing Images activities. Some students will find the flags activity more engaging and creative, while others are itching to see their games take shape. Choose whichever activity is best-suited to you class.*
- *Have students open their game files, and click Run. They should see a frozen screenshot of their game, using the images they requested. (By now, you should have students' graphics already created, and added to the file).*
- So far, everything that you've been doing has been down in the Interactions window. What happens when you click Run at the top? All the work you did disappears!
- That's because the interactions window is meant just for trying things out. If you want to define something permanently, you need to use the Definitions window.
- This is a bare-bones, totally broken game. It doesn't DO anything...YET!
- Look below Step 0, near the top of the screen. Raise your hand if you can read the line of code just below that. *(Have a volunteer read it aloud.)* Raise your hand if you want to take a guess at what will happen if I type `TITLE` into the Interactions window down at the bottom? *Let kids volunteer a few guesses*. Try it out!
- This code tells the computer that the *name* `TITLE` is a shortcut for the string `"My Game"`. When you click Run, the computer learns that name, that shortcut, along with any other definitions.
- When you click Run, you'll see the title "My Game" at the top left hand corner of your new game.
- This kind of name, that's just a shortcut for some value, like a Number, String, or Image, is also called a *variable*. You've seen other names too, like `+` and `string-length` that are the names of functions. You'll name your own functions soon.
- Change the value of this variable from "My Game" to YOUR title. Then click Run, and see if the game is updated to reflect the change.
- Raise your hand if you can tell me the name of the NEXT variable defined in this file *(`TITLE-COLOR`)*. What is its value? *(The string `"white"`)* Try changing this value and clicking Run, until your title looks the way you want it to.
- For practice, try defining a new variable called "author", and set its value to be the string containing your names. Don't forget - all strings are in quotes! *(this won't do anything in the game, but when you close the game window, you can type `author` and see its value. Then you can ask `(string-length author)`, etc.)*
- Raise your hand if you can tell me the name of any other variables you see defined in this file. *Take a volunteer*. What is its name? What is its value?
- Variables can be more that just strings. They can be numbers, or images! These definitions are where we define the images for your background, player, target, and danger.

Game Screenshots — Time: 30 minutes

- Suppose we wanted to combine your game images and layer them together to form a screenshot, so you could see what your game will look like. We want to take these images and stack them on top of each other - what image goes on the bottom? *The BACKGROUND*.
- There's also a variable called SCREENSHOT. What is it defined to be?
  `(define SCREENSHOT (put-image PLAYER 320 240 BACKGROUND))`
- `put-image` is a function that puts one image on top of another, at whatever coordinates you

specify. In our screenshot, what is the image going on the top? *PLAYER*. At what coordinates? *(320, 240)*.

- Try evaluating SCREENSHOT in the interactions window. You should see the player, right in the middle of the background. How would you change the code so the player is a little lower down? To the left? The right? Try it out.
- Now we want to add another image. How about we add the TARGET? Raise your hand if you can tell me what function will let us place this image on top of our stack? It's the one we just used...

```
(put-image  _______
            _______
            _______
            _______)
```

- What image goes on the top? Yes, TARGET! And where should we put it? How about someplace on the right-hand side of the screen? Raise your hand if you can give me some coordinates for that.

```
(put-image  TARGET
            550
            100
            _______)
```

- Now for the final part - what are we putting the TARGET on top of? It's not the background...it's the player and background stack we made earlier! Let's take all that code we wrote, and stick it in as the last input to put-image. See how it almost makes a "staircase" shape? Don't forget to match the parentheses!

```
(put-image  TARGET
            550
            100
            (put-image  PLAYER
                        320
                        240
                        BACKGROUND))
```

- Click "Run", and evaluate SCREENSHOT. Does it look the way you expected? On your own, mess with the coordinates until the TARGET is placed where you want it to be.
- Can you add the DANGER on your own?

Fast Functions — Time: 10 minutes

- You've learned how to write complex expressions, and define shortcuts so you can use them later. That's terrific...but we need more.
- The problem is that all of these expressions always return the same thing - your screenshot, for example, will always look the same, every single time you evaluate it. What you want is a shortcut to a pattern. Then we just fill in the blanks later for the stuff that's changed, and get the whole expression back. Up to now, you've been defining values. Now you're going to learn how to define functions.
- My favorite shape in the whole world is a triangle, and my favorite color is green. I LOVE making solid green triangles! But right now, I have to type out so much code to do that! I need to write the `triangle` function, then a size, then tell the computer that I want it solid and green. I wish there was a shorter way, another function called `gt` that would just take in the size and draw me my triangle.
- Who can help me, by acting out gt? *Take a volunteer*.
- Okay, your name is now **`gt`**. All I need to do is call out your name, give you a number, and you will draw me that beautiful triangle. Let's do a test: "gt fifty!". *The student should draw a solid green triangle on the board*. "gt one hundred!"*The student should draw a solid green triangle, roughly twice as big*.. Can anyone else in the class call this function, passing in a different size?
- Open your workbooks to **page 7**, where it says "fast functions."
- On this page, there is space to write four simple functions. We're going to do the first one together, and then we'll have a competition for the rest.
- Let's start with the contract. What are the three parts of a contract?
- Hey volunteer, what did I say your name was? *gt!* And what information did you need from me to do your job? *just a number - the size!*. And what did you produce, once I'd given you that number? *An Image*.
- Fill in the first contract on the page -- it's the one with the shaded, gray bar.

- Now we have some space to write examples. Let's think about the examples we saw our volunteer act out...
-
- When I wanted him to make a soldi green triangle of size fifty, what did I tell him? "*gt fifty!*". So in the first part of the EXAMPLE, we can write `(gt 50)`. So my example so far is

```
; gt : Number -> Image
(EXAMPLE (gt 50) _____________________________)
```

- Then what did he draw for me? A solid green triangle of size fifty! How would we write the code to draw that same shape?

```
; gt : Number -> Image
(EXAMPLE (gt 50) (triangle 50 "solid" "green"))
```

- Can someone write another example for me?
- Now, on your own, fill out two examples for `gt` on your Fast Functions worksheet.
- If only we had a function like `gt`! Well, let's build one!
- Right now, I'm telling the computer how to deal with a shortcut for "gt 17" - but what if I wanted the shortcut to work for ALL sizes, not just size 17?
- That's the final step: replace the stuff that changes between examples with a variable. So let's look at these two lines, and circle everything that changes. What did we circle? Just the numbers 10 and 17! What do those numbers *mean*? Is it the number of circles we're drawing? No! It's the SIZE. So let's make a little note to ourselves, to remind us that those numbers mean the size of the circle.
- Now we can write the code -- instead of an `EXAMPLE` we'll use `define`. After that, we're just going to copy everything from our examples except the stuff that we circled. What do you think we'll write instead? We'll use the name we wrote down: `size`. *Go character-by-character with the students, asking them if both examples have an open paren, the name "gt", etc...*

```
; gt : Number -> Image
(EXAMPLE (gt   50) (triangle   50 "solid" "green"))
(EXAMPLE (gt   95) (triangle   95 "solid" "green"))
(define  (gt size) (triangle size "solid" "green"))
```

Blue Circle — Time: 10 minutes

- Now it's your turn!
- Raise your hand if you want to help me act out this next function. We'll come up with some examples together, and your group will have to write two more on paper!
- *Hand the student the sign that says "bc" and ask them to come to the whiteboard.*
- When I say "bc 50", you'll draw a solid blue circle of size 50. Let's try it out. "bc fifty!". *Wait for student to draw a circle. Then have several other students give examples to your function, by calling out "bc" and a number. Make sure that the student answering gives an appropriately sized circle return every time.*
- I want to write a function called "bc", which takes in a number and draws me a solid, blue circle that is whatever size the number was. Just like our volunteer here.
- First, you need to write down the CONTRACT for this function. Once again, everyone in your group needs to have the correct answer! You'll have 2 minutes. GO!
- Now it's time to write some examples. Let's look at the first example "bc" drew on the board, for `(bc 50)`. What shape did they draw? What color? What size? How would you write the code to draw that shape? *(write on the board)*:

```
(EXAMPLE (bc 50) (circle 50 "solid" "blue"))
```

- You have 2 minutes for EVERYONE in your group to write out 2 examples of your own. ALL OF THEM have to be correct for your team to get this point. GO!
- Countdown: 30... 10... 5... 4... 3... 2... 1... PENCILS DOWN, EYES ON ME. (Don't forget to wait for total silence, attention.)
- *Give points, praise kids for neat handwriting and good teamwork.*
- *Give the countdown, then review answers with the class and assign points.*
- Time for the last part: writing the function header and body. Your team will have 2 minutes to complete this. GO!
- *Give the countdown, then review answers with the class and assign points.*

Double — Time: 10 minutes

- I want a volunteer to be a function called "double", which takes in a number and multiplies it by two. *Hand the sign to the student*. So if I say "double 3", what will I get back?
- *Have a couple of students try out the function by giving examples*
- You will have TWO minutes to write down that contract and two examples-once you've got your examples, RAISE YOUR HAND and call me over, so I can check them. Two minutes, ready - go!
- *Give the countdown, then review answers with the class and assign points.*
- Raise your hand if you think you know how you could write an example for "double". (*If you get blank stares, give them ONE example on the board. Otherwise, smile and move on.*)
- Your groups will now have FIVE minutes to write two examples, and then circle and label what has changed. Then you can fill out the function header and body. Once you've got your examples, RAISE YOUR HAND and call me over, so I can check them. Do NOT go on to the function header and body until I have checked your examples! Any questions? GO!
- *Give the countdown, then review answers with the class and assign points. If time allows, do another example, preferably one where the domain is something besides numbers.*

Closing — Time: 5 minutes

- Who can tell us one thing we learned today?
- Who saw someone else in the class do something great?
- Well done! You guys have officially started your games! The next step is to make your characters animate, which we'll be doing in our next class. See you then!
- Cleanup, dismissal.