Unit 6
Booleans: Teaching Functions to Compare

Unit Overview

Students discover Boolean types, and use them to create programs that test values, and then model scenarios using these programs.

Agenda

15 min	Introduction
5 min	Booleans
25 min	onscreen? 1.0
15 min	Boolean Functions
25 min	onscreen? 2.0
5 min	Closing

Learning Objectives:

- Learn to work as a team
- Understand how to declare Boolean values
- Learn functions to generate and manipulate Booleans and Strings

Product Outcomes:

- Students will write functions that use conditionals and Booleans
- Students write functions to perform tests on various domains

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.
- Computers w/DrRacket or WeScheme
- Pens/pencils for the students, fresh whiteboard markers for teachers
- Class posters (List of rules, language table, course calendar)

Preparation:

- Write agenda on board
- Student games open on their computers
- "Cage" [Cage.rkt from source-files.zip | WeScheme] preloaded on students' machines, in front
- Class posters
- Seating arrangements: ideally clusters of desks/tables

Language Table

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

Introduction (Time 15 minutes)

- Review the previous material

Booleans (Time 5 minutes)

-  Video Lesson: Intro to Booleans

- We're going to play a quick game, called "True or False."
  I'm going to make some statements, and I want you to call out "True" or "False", depending on whether or not I'm lying.
- *Give some simple boolean statements such as "I am holding a red pen", "I am a boy" etc.*
- So far, our language can only add, subtract, multiply and divide numbers. It can't compare them or decide whether something is true or false! We'll be adding the power of comparison to our language today.
- What would DrRacket say is the value of the following pieces of code?
  `(+ 1 4), (/ 4 2), (- 0 9), (< 3 4)`
- With the exception of the last one, all the above are Numbers. We can solve any of those expressions using our language. Why isn't the last one a number? What is it?
- The expression `(< 3 4)` is a test. It is either true or false that 3 is less than 4. `<` is another function, just like + and string-length. Can you guess what other tests are available?

- Exercise:

  Try using them to compare numbers. What values do you get back? (Use Circle of Evaluation)

```
(> 0 5)
(= 1 9)
(<= 2 2)
(string=? "dog" "cat")
```

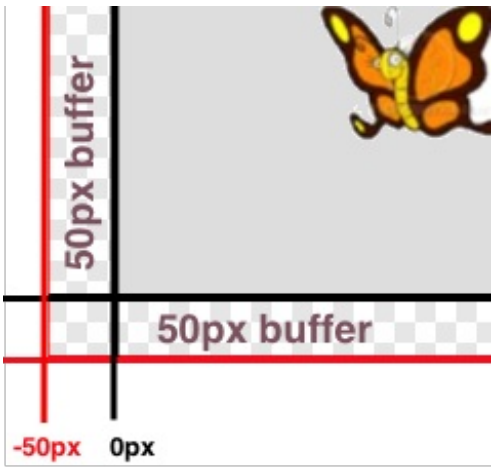
- *Have students explain what each of these tests does.*
- As you have found, there are only two values that can be returned from a test: true or false. These values are a very special category, called "Booleans". Let there be Booleans!
- *Model one of the contracts for a Boolean function, then have them guess the rest.*
- Try filling out all of the contracts for these in your notes. For example: `; Number Number -> Boolean`
- Booleans are really important in videogames - you can use them to test if someone's health is down to zero, or if they've bumped into a wall. Without booleans, there are no tests, and the game can't ever react to changes.
- Let's try making a function that uses booleans.

onscreen? 1.0 (Time 25 minutes)

- *Have students open the Butterfly File.*
- When you click Run, a window with a butterfly in it will pop up. You can use the arrow keys to move the butterfly around.
- This butterfly is Sam - he's a happy student like you! He likes to fly around outside and enjoy the fresh air.
- His yard is 640 pixels wide and 480 pixels tall. If the bottom-left corner is (0, 0), what are the coordinates of the other corners?
- Sam's mom tells him that he should stay where she can see him - that means he can even step a little outside of the yard...as long as a piece of him is visible, his mom is okay.
- Try moving him so he's a little off the screen, but still safe. How far can he go?
- Sometimes, however, Sam sneaks out of the box and just keeps going! He's free! Free as a bird! (well, as a butterfly!) Look at his smile! He's so happy to be free! Why shouldn't he be?
- Well...there's some bad news.
- Sam doesn't realize that there's a monster outside his yard, waiting to eat him! As long as ANY part of Sam is still in the yard, he is safe...but the moment he disappears completely his mom won't be able to keep an eye on him!
- It's up to you to keep him safe!

- *Draw and label a 640x480 rectangle on the board to represent the screen, and have students help you idenfify the coordinates at the corners. Then turn students' attention to the computers, and have them analyze the last function in the code, called* `onscreen?`.
- This file contains three function definitions. Can you find all three? What are the names of these functions? *(*`safe-left?`, `safe-right?` *and* `onscreen?`*)*
- *Skit:*

  Have a volunteer stand up to be `safe-left?`*. Ask them about their domain and range, and what they do according to the code (they always return true). Have students practice calling the function with sample values, then have another volunteer repeat the process as* `safe-right?`*. Finally, have a third volunteer do the same as* `onscreen?`*. Note that* `onscreen?` *must use* `safe-left?` *when evaluate an input!*



- The computer only talks to `onscreen?`, when deciding whether or not to let Sam keep moving. In turn, `onscreen?` will need to use both `safe-left?` and `safe-right?` to figure out if Sam is safe on both sides.
- What are the Domain and Range of `onscreen?` What does the function do? What will happen when it's given a sample input? (Answer: It calls the function 'safe-left?' on the variable x)
- We need to fix this code, to keep Sam safe. He can fly left or right and there is nothing to stop him from moving off the screen... yet!
- How far can Sam go to the left before he disappears completely? (Answer: -50.) So he is onscreen as long as *x is greater than -50*!
- *Take some ideas from the class, then explain if necessary.*
  When the butterfly is drawn at some coordinate, it is centered at that coordinate. So if it's drawn at 0, half of the butterfly is actually off the screen. We add the extra 50 pixels on all sides, because a piece of him is still visible as long as he is within 50 pixels of the screen's edge.
- Turn to Page 17. Take thirty seconds to fill out what we've discovered.
- Turn to Page 18 and read the word problem for the `safe-left` function.
- *Who would like to act out* `safe-left`*? Take a volunteer.*
- *Exercise:*

  According to the word problem, what's your name? What's your Domain? Your Range? Given Sam's x-coordinate, you need to tell me if any part of him is still on the screen. Let's try it out. "safe-left forty-five!" "safe-left three!" "safe-left negative sixty-one!" How did you know Sam was not protected at -51? because the number was not greater than -50!.

- Complete the design recipe for `safe-left`.
  Raise your hand after you complete each step.
- When a team has completed the Design Recipe for `safe-left`, they must type in the examples and function body onto the computer.
- So now we have a function that will tell us whether or not Sam is safe on the left side of the screen.
- Click "Run", and try to move Sam off the left edge of the screen. Congrats! You've protected Sam on one side!
- Unfortunately, Sam can still escape on the right hand side. We need a `safe-right`!
  (act out with another volunteer, if necessary).
- Turn to Page 19, and write another function called `safe-right`.
  Call me over when you reach a stopping point!
- Notice that even though we've fixed `safe-right?`, `onscreen?` won't know about it until we change the body of `onscreen?` so it calls `safe-right?` instead of `safe-left?`! Please make this change now, so that Same is safe on the right side instead of the left side.
- Good job! Now we can keep Sam safe on the left side, or we can keep Sam safe on the right side. However, we don't have a way to keep him safe on both sides! When we call `onscreen?` with an x-coordinate, what should it do?

Boolean Functions (Time 15 minutes)

- You've already learned many functions that allow you to create Booleans. < and >, for example, will all return a boolean when applied to the appropriate input.
- There are also many functions that work with Booleans, and let you do really complex tests.
- Tell me if the following statements are true or false:

    - Sugar is sweet.
    - Ice is hot.
    - Sugar is sweet **and** Ice is cold.
    - Sugar is sweet **and** Ice is hot. Why is it false? Isn't sugar still sweet?
    - Sugar is sweet **or** Ice is cold.
    - Sugar is sweet **or** Ice is hot. Why is it true? Ice isn't hot!
- Did you notice how we joined the Boolean statements? What words did we use? (AND, OR).
- The key concept here is that `and` requires that both statements be true, and `or` needs *at least one* to be true.
- What are the contracts for both AND and OR? (Answer:

```
; and: Boolean Boolean -> Boolean
; Returns true if BOTH inputs are true

; or: Boolean Boolean -> Boolean
; Returns true if EITHER of the inputs are true
```

)
- Turn to Page 20 and read the directions
- Take five minutes with your team, to draw the circle for each of the statements. Think about what Boolean functions you'll need for each one! When you're done, convert those circles into Racket code.

onscreen? 2.0 (Time 25 minutes)

- Now it's time to put it all together. Right now, you know how to keep Sam safe on the left side of the screen, and how to keep him safe on the right side. But how do we keep him safe on BOTH sides at the same time?
- Turn to Page 21 and read the word problem.
- *Have the `onscreen?` volunteer stand again.*
- What is your name? What is your domain? What is your Range?
- Sam is onscreen only when he is safe on the left AND the right. How can we use one of those functions we just learned to do this?
  'd like my volunteers `safe-left?` and `safe-right?` to stand up again.
- *When I yell out "onscreen? fifty", `onscreen?` is going to call `safe-left?` and `safe-right?` with that number. Depending on what they give back, `onscreen?` will tell me true or false*
- *Try this out, making sure that students are actually talking to one another, calling the functions properly by name. When you call out "onscreen? eighty!", the `onscreen?` volunteer should call out "safe-left? eighty" and "safe-right? eighty", and listen their answers before giving back an answer of their own. Run through a bunch of examples here, to make sure it's clear.*
- It's time to write the contract and examples! Don't forget what we just went over!
  Raise your hands so I can check your work at each step
- *Guide students as necessary, and eventually have them write the code on the computer.*
- Guess what? These functions are exactly what we need in our game file, and to have your characters come back once they've gone off the screen!
- Copy and paste the three functions `safe-left?`, `safe-right?`, and `onscreen?` into your game files.
- There are two ways to go about solving this word problem:

```
(define (onscreen? x)
(and (> x -50) (< x 690)))
```

  and...

```
(define (onscreen? x)
(and (safe-left? x) (safe-right? x)))
```

- Does it matter? Both will work just fine when you type them in...so why care?
- There's more to being a writer than putting down grammatically-correct sentences. There's more to being an architect or an artist than building a bridge or coloring in a canvas. All of these disciplines involved an element of *design*.
- Likewise, there is more to being a Programmer than just writing code. You've gotten decent at writing code, and now you're able to write sophisticated programs that include multiple functions - congratulations! But that also means you're ready to consider what it means to *design* the code that you write.
- The first solution puts all of the logic into one function: the left boundary and the right boundary are both tested by `onscreen?`. In the second solution, `onscreen?` uses both helper functions, each of which only tests one boundary

at a time. Can you think of a reason why one might be better than another? Is it better to have a single, complex function that does all the work, or to have multiple simple functions that only do one thing apiece?

- Suppose you just built a car, but it's not working right. What would you do? Ideally, you'd like to test each part of the car (the engine, the transmission, etc) *one at a time*, to see which one was broken. The same is true for code! If you have a bug, it's much easier to find when every function is simple and easy to test, and the only complex functions are just built out of simpler ones.
- Another reason to like the second example is the fact that it lets programmers be lazy. Suppose you have a few characters in a videogame, all of which need to be kept on the screen. Some of them might only need `safe-left?`, others might only need `safe-right?`, and only a few might need `onscreen?`. What happens if the game suddenly needs to run on computers with differently-sized monitors, where the boundary is 1000 instead of 690? If you have simple and complex functions spread throughout your code, you'll need to change them all. If your complex functions just use the simpler ones, you'd only need to change them in one place!
- Badly designed programs can work just fine, but they are hard to read, hard to test, and easy to screw up if things change. As you grow and develop as a programmer, you'll need to think beyond just "making code work". It's not good enough if it just works - as artists, we should care about whether or not code is *well designed*, too.
- This is what functions allow us to do! Everyone from programmers to mathematicians uses functions to carve up complex problems into simpler pieces, which make it possible to design elegant solutions to difficult problems.
- So the next time you sit down to solve a problem, think about algebraic abstraction, and how you might design a beautiful solution out of simple functions, rather than a huge, complex one.

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.*