



Unit 7

Conditional Branching

Unit Overview

Students use geometry and knowledge of basic image functions to design characters for their games, this time using conditional branching to accommodate different key-events.

Learning Objectives

Students will:

- Reason about the relative positioning of objects using mathematics
- Discover Partial Functions, and how to implement them using `Cond`
- Use Booleans with `cond` to change control flow
- Adapt Design Recipe to add `cond`

Product Outcomes

- Students will write functions that use conditionals and Booleans
- Students will write `update-player`

State Standards

See [Bootstrap Standards Matrix](#) provided as part of the Bootstrap curriculum.

Length: 90min

Materials and Equipment

- Computers w/DrRacket or WeScheme
- Student [workbooks](#)
- Class posters (rules, basic skills, calendar, warnings, Design Recipe, language table)
- Pens or pencils for students

Preparation

- Write agenda on board
- All student computers should have their game templates pre-loaded, with their image files linked in
- Seating arrangements: ideally clusters of desks/tables

Language Table

Range	Functions
Number	<code>+ - * / sqrt expt string-length</code>
String	<code>string-append</code>
Image	<code>rectangle circle triangle ellipse radial-star scale rotate put-image</code>
Boolean	<code>= > < string=? and or</code>

Agenda

- 10min [Introduction](#)
- 30min [Pizza Toppings](#)
- 25min [Player Movement](#)
- 5min [Closing](#)

Introduction**Time: 10 minutes**

- Review previous material.

Pizza Toppings **Time: 30 minutes**

- Turn to the Design Recipe on **Page 23** and grab a Design Recipe Worksheet.
- Suppose we've been hired by Luigi's Pizza to write a function that tells us the cost of different pizza pies. Let's use the design recipe to write this function. Have a student read the problem statement.
- I need a volunteer to be our function. *Pick someone, and copy the contract as they answer.* What is your name? *cost* Your Domain? *String* Your Range? *Number*.

```
; cost: String -> Number
```

- Can someone from the class tell me how we should call this function? For example, "cost 'cheese'!" What will `cost` produce? Let's try this with other toppings...
- Now it's time to write down some examples. Can anyone raise their hands and tell me what I'd write?

```
(EXAMPLE (cost "cheese") 9.00)
```

- What are some other examples for `cost`? What changes between them? The **topping** and the **price** returned! Make sure you label those.
- Do you notice something odd here? This is the first time that we've ever circled something in the second of the examples, which wasn't also circled in the first part. The price that's being produced changes, but the function never takes in the price!
- That's a hint that something special is going on, but let's see how much farther the Design Recipe can take us...
- Now for the Function Header. What do I write here? (`define (cost topping)`)
- The Function Body is next. But now we don't know what to write! We know that our examples behave differently from one another -- sometimes we want to return 9.00, other times it's 10.50, etc. So what do we do? Well, we could fill in *all* off those results. Let's do that...*Make a large, 2-column table on under the Function Header.*

```
(define (cost topping)
```

	10.50
	9.00
	11.25
	10.25

```
)
```

- But how do we know when we want to produce 9.00? When the `topping` is "cheese". When do we want to return 10.50? When the `topping` is "cheese".
- What we want is a way to go down each line, checking to see if the topping is the right one. If it is, we go on to finish the line. If not, we go on to the next one.
 - What's Domain of our function? String (according to the contract)
 - What's the type of "pepperoni"? *String*
 - What function compares two strings, and gives back a Boolean? `string=?`
 - What's the Racket code that compares the input topping to the string "pepperoni"? (`string=? topping "pepperoni"`)
 - Now we can write that on our first line, as our first topping check. Can you do the rest?

- Have the students fill in the rest of the table...

```
(define (cost topping)
```

<code>(string=? topping "pepperoni")</code>	10.50
<code>(string=? topping "cheese")</code>	9.00
<code>(string=? topping "chicken")</code>	11.25

```
(string=? topping "broccoli") 10.25
```

```
)
```

- Each of these rows is called a *condition*. A condition has a test and a result. The computer goes down the code, one condition at a time, and will evaluate the first result for which the condition is true.
- Racket has a special function that lets us tell the computer to do this: `cond`. To use `cond`, you put square brackets around each of the branches, and write "cond" at the top:

```
(define (cost topping)
  (cond
    [(string=? topping "pepperoni") 10.50]
    [(string=? topping "cheese") 9.00]
    [(string=? topping "chicken") 11.25]
    [(string=? topping "broccoli") 10.25]
  )
)
```

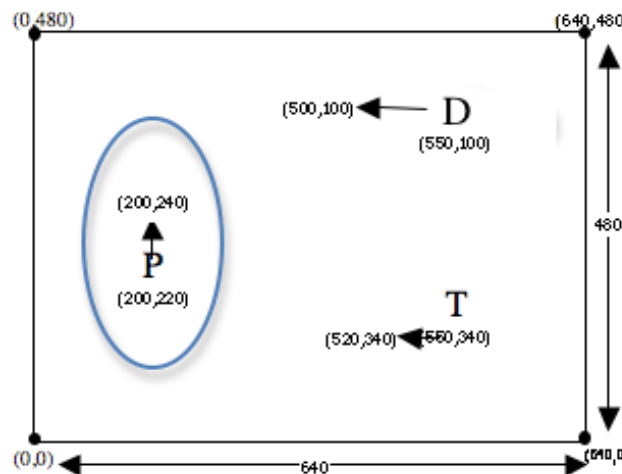
- Remind students that computers are very specific and can't make up new answers; we need to tell it what to do in case the user inputs an item that is not in our list. Let's add `else`. If it's not on the menu, we might still make that pizza for you, but it'll cost you! [`else 10000000`]
- Have students try it on the computers, adding new items on their own.
- If you have additional time, and would like to try another `Cond` challenge, check out the [supplemental activity](#).

Player Movement

Time: 25 minutes

- Great! Now that we know `cond` we can write `update-player`.
- Draw a screen on the board, and label the coordinates for a player, target and danger. Circle all the data associated with the Player.

- What is the player's starting x-coordinate?
- It's starting y-coordinate?
- What about after it moves? What's the new x and y? What has changed? And by how much? What happens when we press the down key? What should the new coordinates be then?
- Get students to tell you what `update player` should do...
- We want a function that will move up the screen when the user presses the up arrow and down when the user presses the down arrow.



- We've set up the computer to call `update-player`, passing in the player's y-coordinate and the name of the key pressed. The keypress will either be the string "down" or the string "up" (for now). What kind of data is the y-coordinate? What kind of data is the keypress?
- Make a table showing possibilities and results, walking students through it.
- With our pizza example, we had to deal with toppings that weren't on the menu. Now we need to deal with keys that aren't "up" or "down". How do we do that?
- On **page 24**, you'll find the problem statement for `update-player`. Grab a Design Recipe Worksheet, fill it out, and then write this function with your team.
- Students can also add "cheat codes", by adding `Cond` branches for other keys. For example, a student might add [`(string=? key "c") 240`], which causes the player to jump to the center of the screen if the `c` key is pressed.

Closing

Time: 5 minutes

- Congratulations - you've got the beginnings of a working game!
- What's still missing? Nothing happens when the player collides with the object or target!
- We're going to fix these over the next few lessons, and also work on the artwork and story for our games, so stay tuned!

- Who can tell us one thing we learned today? Call on 2-3 volunteers.
- Who saw someone else in the class do something great? Call on 2-3 volunteers.
- Cleanup, dismissal.

Bootstrap by [Emmanuel Schanzer](#) is licensed under a [Creative Commons 3.0 Unported License](#). Based on a work at www.BootstrapWorld.org. Permissions beyond the scope of this license may be available at schanzer@BootstrapWorld.org.

