Don't Dictate, Evaluate! An Introduction to Functional Programming with Dr. Racket

June 20, 2015

Part I
Introduction

1 Ground Rules Reminder

- You must be respectful and courteous toward everyone in this room.
- You must keep your attention focused on functional programming with Dr. Racket.
- Call me "mulhern" or "Dr. Mulhern" if you like formality.

2 Expectations

You will not write an amazing application today, but you will achieve some pretty neat visual effects. You will learn about a way of programming that is a little different from what is most normal or typical and get a chance to experiment with it with some guidance. You will not get a complete introduction to Dr. Racket or functional programming, because this tutorial is only two hours and two hours is not enough for a complete introduction. But I hope the experience will be enough to open your minds to some pretty subtle new ideas and to a hugely valuable way of thinking and programming.

3 Getting the Most Out of this Tutorial

This tutorial is not a cookbook-style tutorial where you will follow the same directions as everybody else and, if you get it right, end up with the same program at the end. Simple Dr. Racket programs that do neat things are very easy to write. So I will focus on teaching you just enough so that you can experiment with existing programs, and come up with new and interesting variations on these programs. The format of the tutorial will be something like the following.

First, I will show you a few things. Do not feel that you have to follow along while I show you, I have tried to put everything you need to know in this document. After a short time, I'll stop and ask you to copy what I've done, and then experiment a little on your own. If you run into trouble, ask someone sitting near you for help. If nobody near you can help, ask me. Make sure to try out a few different things. Eventually, I will ask you all to stop, and I'll explain to you what you have done. Then we'll go through the whole cycle again a few more times.

There are some sections marked "Extra". You do not need to do those to do the next part of the tutorial, but you should feel free to do them if you have extra time.

Do not try to copy and paste images or example Dr. Racket expressions from this document. Type them in instead. The Dr. Racket expressions in the document do not copy well to Dr. Racket and you will do better if you get the practice of typing them, anyway.

Make sure to save your files to your USB key before you move on to the next step.

Part II
Let's Get Started

4 Getting Started

4.1 Setup

Start Dr. Racket and make sure that your environment is set up correctly.

1. Start a new file.
2. For your language, select "Intermediate Student with Lambda".

Hint: If you see a yellow warning bar at the bottom your changes have not yet been applied. Press the "Run" button to apply your changes.

4.2 Loading the Racket Documentation into Your Browser

To load the documentation, choose "Help" and then "Racket Documentation". Your browser will open to display the documentation on your laptop. After the documentation page is done loading, look for "How to Design Programs Teachpacks" and then for the "2e" teachpacks specifically.

4.3 Entering Values

Enter some *values* at the prompt. Press the return key after you enter each value.

1. The easiest values to enter are numbers, e.g., 1, -3, 2.14, 3/5, 1e10, 1+3i. The last two numbers are slightly unusual. The first is a number represented using scientific notation and the second is a complex number, but they are both just numbers. You can enter all these numbers just by typing at the keyboard.

2. Next easiest are images. See the Racket Planet Cute Images documentation for a few simple images. You can enter these images just by copying the image and then pasting it at the prompt.

Make sure that you are entering your code in the "Interactions Pane". If there gets to be too much clutter on your screen, just press the "Run" button to clear it.

Note that the *intepreter* responds, every time, by displaying the value that you entered. If you entered multiple values before pressing the return key it displays one value, then the next, and so on.

4.3.1 Extra!

Next easiest are strings. Strings are just lists of characters. However, Dr. Racket needs some extra information to distinguish a string from other things. So that Dr. Racket can tell that you mean a list of characters, you must surround your list of characters with a pair of double quotation marks, so

```
"I am an example string"
```

means to Dr. Racket an *I*, followed by a space, followed by an *a*, then an *m*, and so forth. Dr. Racket displays a string value with surrounding quotation marks, so that you will understand that the value is a string.

5 Next Steps: Programming, i.e, evaluating

5.1 Importing Some Useful Libraries

Add the following additional lines at the top of your Definitions Pane.

```
(require 2htdp/image)
(require 2htdp/universe)
(require 2htdp/planetcute)
```

These are all libraries of useful names that you will find convenient.

5.2 Entering Non-Value Expressions

Enter some non-value expressions at the prompt. Pay most attention to expressions that evaluate to images. Note that expressions that are not values are always surrounded with parentheses.

5.2.1 Expressions where the value is a number

You can do the usual arithmetic computations, like adding, subtracting, and so forth. However, Dr. Racket expects a different *syntax* than you may be used to. All arithmetic expression follow an obvious pattern.

- (+ 2 32) means the sum of 2 and 32
- (∗ 2 32) means the product of 2 and 32
- (/ 2 32) means 2 divided by 32
- (remainder 37 14) means the amount that is left over after 14 is subtracted from 37 as many times as possible, 9

The remainder operator is often not so important in your mathematics classes, but is very handy in the writing of computer programs.

5.2.2 Expressions where the value is an image

There are lots of expressions where the value is an image.

- (circle 32 "solid" "red") evaluates to an image: a red, filled in, circle with radius 32
- (square 32 "outline" "blue") evaluates to a square with some different properties

You can construct many more shapes than this. The Racket Documentation describes them all.

Besides circle and square, radial-star makes some particularly nice images, but star, rhombus, rectangle, ellipse, and triangle are also nice. Try out different colors as well. Dr. Racket knows about magenta, lavender, and crimson, but not puce.

5.2.3 Extra!

You may already know about the RGB representation of colors. An expression that evaluates to a color can be constructed with the *make-color* function. For example,

```
(make-color 255 0 0)
```

makes the color red, because it gives the maximum possible value to the red component of the color and the value 0 to the other two components. So the expression

```
(circle 30 "solid" (make-color 255 0 0))
```

evaluates to exactly the same value as the expression

```
(circle 30 "solid" "red")
```

. You can make expressions with make-color using three numbers or four numbers. If you use four numbers, the fourth number represents opacity.

Note that the representation of the value of the expression is just the expression itself; when there is no obvious better way to represent a value, Dr. Racket chooses the simplest expression that constructs the value.

The best way to see what the color looks like is to make a shape of that color.

6 Making Complex Expressions

This section is about building up more complex expressions from simple expression.

6.1 Making Your First Complex Shape

Open the file "complex-shapes' in Dr. Racket. You will find this file in the examples folder (./examples). Press the "Run" button. At the prompt enter my-star and my-circle. The interpreter will display a star and a circle. Enter the expression

```
(overlay my-star my-circle)
```

. The value of this expression is a star overlayed on a circle. Enter

```
(scale 10 my-star)
```

. The value of this expression is a star ten times bigger than the value of my-star.

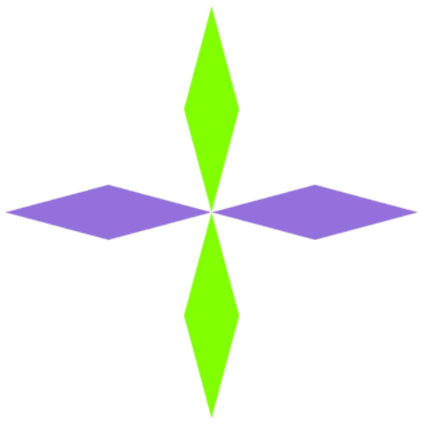
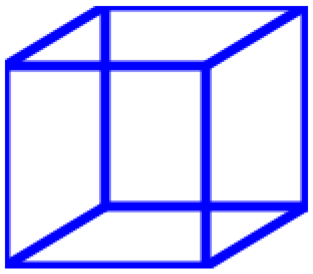
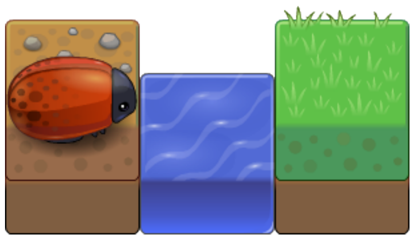
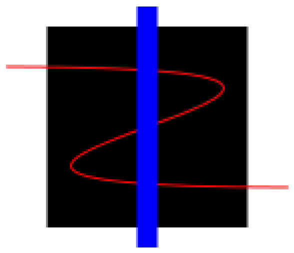
6.2 Anatomy of the Definitions File

There are only two lines in the file.Each of these lines is a *definition.* A definition assigns a value to a name. Each definition starts with *define*, then a name, and then an expression. It is the *value* of the expression, not the expression itself, that the name stands for. Wherever you need to, you can always use the name, my-star, in place of the star image that it stands for.

6.3 More Expressions and More Definitions

Experiment with the other ways of writing complex image expressions in Dr. Racket. The Dr. Racket Documentation describes them, with examples. Besides overlay, beside and above are quite nice for combining multiple images. Besides scale, rotate, flip-horizontal, and flip-vertical are nice ways for writing expressions with a single image.

Here are a few simple examples to get you started.

For working with Planet Cute images, overlay/xy and beside/align are often helpful.

6.4 Extra!

- Experiment with textual images.

  ```
  (text "Hello" 32 "blue")
  ```

  evaluates to a blue string of moderate size. You can write expressions with textual images in the same way that you can write expressions with any other kind of image.

- Experiment with using make-color when writing image expressions.

7 The Really Important Point

In a functional programming language *every expression* has a value.[1] Contrast with Scratch (and many other languages) where some expressions have a value but quite as many do not.

[1]Well, not unless the language is *purely* functional, like Haskell. But we are using a functional core of racket, so every expression that you will use today will evaluate to a value.

Part III

Project: A Simple Animation

This part of the workshop is about very simple animations and the essential concept of *functions*.

8 Getting Started

8.1 Running Your First Animation

Open the file "expanding-circle.rkt" in Dr. Racket. You will find this file in the examples folder (./examples) of this handout directory. Press the "Run" button. At the prompt, enter

```
(animate circle-on-background)
```

and press the return key. This animation runs forever. When you become tired of it, close it. Notice that when you close it, the interpreter displays a number; clearly the value of the expression is a number.

8.2 The Anatomy of Your First Animation

Look at the code in the file. There are only 5 distinct items . Three of the blocks start with *define*, and two with *check-expect*.

The expression in the first definition is just like the expressions that you have already worked with.

The two other expressions are lambda-expressions. These represent *functions*. Each of the function definitions has a test block below it, which shows how the function is used, and what it is expected to do. Each of the tests starts with check-expect, then an expression which uses the function name, then a different expression which evaluates to the same value as the first expression.

Use the tests as examples that show you how to form expressions with the function names. In each case, the expression should evaluate to an image. Try several different expressions that use the function names. For these functions it turns out that every different expression has a different value.

8.3 The animate function

You probably already know that a movie is nothing but a sequence of still frames. The frames are displayed so rapidly that your brain interprets the sequence of images it sees as continuous motion. The animate function works just like a movie. The frames it constructs and displays are the values of the expressions

```
(circle-on-background 0)
(circle-on-background 1)
(circle-on-background 2)
```

```
(circle-on-background 3)
(circle-on-background 4)
```

and so forth.

The pattern here is pretty obvious; what changes in the expression is the number, which goes up by 1 every time. Of course, the value of the expression changes accordingly, and the circle on the background gets larger as the number increases.

9 Modifying the Animation

Save your definitions file with a new name, like "my-animation.rkt" and experiment with modifying the definitions to change the animation. Experiment by modifying your definitions in small steps. You can do really simple things, like modifying the color or size of the background. You can make the circle grow more slowly or more rapidly using some very simple arithmetic. You can make the size of the circle fluctuate in interesting ways using somewhat more complex arithmetic. You can change the type of shape that gets displayed. You can use Planet Cute images instead of geometric shapes.

10 Getting Rid of Failing Tests

If you change your animation substantially, your tests will probably fail. If you get rid of the tests, your untested code will be highlighted in yellow/black. Use the "Clear Error Highlights" menu item to clear this, if it bothers you. Eventually, you will wish to change your language to the full racket language. You can do this by selecting the Racket language, and putting the line #lang racket at the top of your file.

11 Extra!

- Use the fact that colors can be made from numbers to write an animation where the color changes instead of or in addition to the size of the circle. Since the maximum value for any of the RGB values is 255, the remainder operator is very useful to prevent your animation from crashing.
- Use overlay/xy or overlay/offset to move a shape on the background.

12 The Really Important Point

In a functional language, functions are values. This means that a function can be treated in exactly the same way as any other value, e.g., 0. This is true of all functional languages, even those that are not purely functional, because it is so very handy.

Part IV
Extra: A Simple GUI Application

13 Getting Started

Open the file "gui-app.rkt" in Dr. Racket. You will find this file in the examples folder (./examples) of this handout directory. Press the "Run" button. At the prompt, enter

```
(my-gui-app circle-on-background)
```

and press the return key. It does exactly the same thing as the expanding circle animation. The big difference is that my-gui-app does not use animate, it uses big-bang. animate is a simple function for making simple animations; big-bang is a really general function for making GUIs. The next thing to do is to extend my-gui-app so that it reacts to key presses.

13.1 The Anatomy of this GUI

13.1.1 Understanding halve

Look at the code in the file. There is only one new definition, halve. Examine its test, and write a second, different one, that passes. There are lots of clues to what this halve function does. First, its name means to cut in half. Second, note the use of the / for division in the body of halve. Third, note that 150 is one half of 300.

13.1.2 Using halve

Modify my-gui-app to use halve. In a reasonable place near the end of the big-bang expression, insert the expression[2]

```
(on-key halve)
```

. I always go in alphabetical order, so I would put the on-key expression before the on-tick expression, but not before the 0. big-bang has around 20 possibiliites besides on-key, on-tick, and to-draw, and alphabetizing them makes it easy for me to find them. Now, start my-gui-app again, and while it is running, press any key. Your GUI will respond.

13.2 Understanding the GUI

This GUI uses a design concept that all properly constructed GUIs use, the MVC design. It has three parts: model, view, and controller. In this example, the model is a number. The initial value of the number is 0. The view is how

[2]This is not actually an expression in the expression/value sense, but it looks just like an expression.

the model is displayed; that is handled by the circle-on-background function. The controller is everything that changes the model. The controller is always an expression that evaluates to a new value of the model using the old value of the model and sometimes other information. In the animation, the only expression was

```
(add1 world)
```

. This expression always evaluates to one more than the value of world.

For this interactive GUI you just added an on-key expression that cause the model to be divided by 2 when any key is pressed.

Now, modify my-gui-app by adding the expression

```
(state true)
```

and run your application again. You will see an additional screen which shows the model, a single number. At first, the number will be 0, then it will increase by 1. When you press a key, it will be halved.

13.3 Modifying the GUI

You can try any of the ideas from the previous animation section to modify your GUI. Remember that your on-key function is part of the controller, and the model is just a single number. So, any expression that starts with your on-key function must evaluate to a number. If it evaluates to anything else, your GUI will crash. Try it and see.

You can change the initial value of the world, and you can change the on-tick function as well.

14 The Really Important Point

Model, View, Controller. Keep them separate. If you fail, confusion abounds.

Part V
Next Steps

15 Some Complete Programs

Look in the subdirectory ./extras for a few complete examples. These all start as soon as you press “Run”.

16 More Resources

Here are a bunch of options that you can try if you want to continue working with Dr. Racket.

- Dr. Racket, along with lots of useful resources, is freely available online at: `http://racket-lang.org/`.
- "Realm of Racket" is a book with a focus on programming games and a website here: `http://www.realmofracket.com/`.
- "How to Design Programs, Second Edition" is a very complete work in progress with a website here: `http://www.ccs.neu.edu/home/matthias/HtDP2e/`. It is about how to design programs in any language, but the authors, who are also the developers of Dr. Racket, make use of the teaching languages and teachpacks you have used today.
- Pyret (`http://www.pyret.org/`) is an exciting spinoff project. You do not need to install anything, you can develop your programs in a browser. Pyret uses indentation, and does not have so many parentheses, so it is arguably easier for a humans to read Pyret code.
- "Picturing Programs" (`http://picturingprograms.com/`) has about the same goals as "How to Design Programs". It has its own graphical teachpack, which has some more capabilities than the standard teachpacks we used in this class.
- This website (`http://www.cdf.toronto.edu/~heap/racket_lectures.html`) has a bunch of instructional videos about programming in Dr. Racket.

17 Really Important Vocabulary

definition A definition is the assignment of a value to a name.

function A function that maps values to values. In mathematics, usually written as $f(x) = x + 1$. In Dr. Racket, the same function would be written as (define (f x) (+ x 1)).

interpreter A program that executes another program in a given language. e.g. a Python interpreter, a Java interpreter, etc.

lambda-expression A function value.

syntax The rules for forming valid expressions in a language.

value A value is an expression that can not be further evaluated to a simpler expression.