v.6.8

Lab 3 Hello World

Purpose: The purpose of this lab is to give you some hands-on experience with designing interactive programs and programs that process structures and itemizations of data.

Textbook references: Chapter 2.5: Programs, Chapter 3: How to Design Programs, Chapter 6: Itemizations and Structures

Chip the Cheap Sheep

Goals: Practice defining constants. Create data and structure definitions. Create main functions for world programs. Create and process wishlists of functions.

An animation studio is working on a game called "Chip, the Cheap Sheep". They have a basic idea, which is to have a sheep running across the screen. Here are some sheep shapes that their artist has drawn:

Your goal is to create a simple proof-of-concept game with Chip running from the right side of the screen to the x-coordinate of the point the player clicks on first. Before Chip runs, the screen shows the text

click the mouse to get Chip running

Sample Problem Sketch some still frames from the game. Which aspects remain the same? Which change and what causes them to change?

Sample Problem Create a table of what changes in the game (part of the world) and what stays the same (constants).

Constants (what stays the same)	Part of the world (what changes)
The size of the screen	Whether or not Chip is running
The photos of Chip	Where Chip is running to
The initial text	Where Chip currently is

The background What photo of Chip to show

Sample Problem You should be able to drag the images from the webpage into DrRacket, and once they are there you can define them as constants. If not, save them to the desktop, and use "Insert">"Insert Image..." from the DrRacket menu bar. You can also copy and paste the images into DrRacket.

Sample Problem Gather constant definitions for the properties of the world frames that remain the same. Here are some:

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

(define WIDTH (* 20 (image-width CHIP1)))
(define HEIGHT (* 2 (image-height CHIP1)))
(define PLAYGROUND (empty-scene WIDTH HEIGHT))
(define TXT (text "click the mouse to get Chip running" 33 "pink"))
```

Sample Problem Develop a data definition so that your program can keep track of all the quantities that change between frames.

Hint: In addition to where Chip currently is, your program must know where Chip is going. After all, at the very beginning you have no clue where Chip is supposed to go. Develop a data definition so that you can represent the state of the game. Remember that data definitions come with an interpretation, sample data, and a template.

```
; A ChipWorld (CW) is one of:
; - "click to get Chip running"
; - (make-running-chip Number Number ChipFrame)
(define-struct running-chip [goal current frame])
; A CW represents whether or not the user has clicked
; to get Chip running.
; If Chip has begun to run:
; - goal represents the x-coordinate to which Chip is running,
; - current represents the current x-coordinate of Chip
; - frame represents which animation frame to show
(define CHIP-WORLD-STILL "click to get Chip running")
(define RUNNING-CHIP (make-running-chip 30 300 0))

; cw-temp : CW -> ?
(define (cw-temp cw)
  (cond [(string? cw) ...]
        [else (... (running-chip-goal cw)
                   (running-chip-current cw)
```

```
                    (cf-temp (running-chip-frame cw)))]))

; A ChipFrame (CF) is one of:
; - 0
; - 1
; - 2
; - 3
; and represents which frame of Chip to show
(define CF-0 0)
(define CF-1 1)
(define CF-2 2)
(define CF-3 3)

; cf-temp : CF -> ?
(define (cf-temp cf)
  (cond [(= cf CF-0) ...]
        [(= cf CF-1) ...]
        [(= cf CF-2) ...]
        [(= cf CF-3) ...]))
```

Sample Problem Develop a wishlist. You know you need a function that renders the current state of the game, as all big-bang programs require this. What else do you need? Which kind of events make the game transition from one state to another?

```
; Wishlist

; ————————————————

; CW changing over time

; move : CW -> CW
; Move chip closer to his goal and select the next ChipFrame (if Chip has begun to run)

; next-frame : CF -> CF
; The next chip frame

; ————————————————

; Drawing CW

; render : CW -> Image
; Present either the initial screen or an in-progress game

; frame->image : CF -> Image
; Select the correct image based on the current frame
```

```
; ————————————————

; Handling the click

; launch : CW Number Number MouseEvent -> CW
; Launch the game if the person clicks and the game has not already been launched

; ————————————————

; Stopping the game

; at-goal? : CW -> Boolean
; Is Chip running and is he close enough to his destination?
```

Exercise 1 Complete the functions in the wishlist. Remember, since we are now using the design recipe, completing a function **must** include tests. They must also include signatures and purpose statements, but one of the advantages of making a wishlist is that you will already have written these.

Exercise 2 Use the following main function to run your world program. Because we wrote a wishlist, there is only one main function we can write (given the following signature and purpose statement, of course). Planning ahead makes writing code much easier!

```
; play-chip : PositiveNumber -> CW
; Given the animation rate, play a game of Chip the cheap sheep
(define (play-chip tick-rate)
  (big-bang CHIP-WORLD-STILL
            [on-tick move tick-rate]
            [to-draw render]
            [on-mouse launch]
            [stop-when at-goal?]))
```

Shrinking Balloons

Goals: Practice defining constants. Create data and structure definitions. Create main functions for world programs. Create and process wishlists of functions.

Now that you've designed and implemented Chip The Cheap Sheep, you will design and implement a simpler game, called Shrinking Ballon. Shrinking Balloon starts with a circle in the center of the screen and shrinks constantly over time. When someone clicks, it grows. Exactly how fast it should grow and shrink is up to you.

The game ends when the balloon is too small to see (what this means is also up to you).

Exercise 3 Create a table of what changes in the balloon game (part of the world) and what stays the same (constants).

Exercise 4 Add appropriate constant definitions to your definitions area.

Exercise 5 Define a structure type for the world and then develop a data definition, examples, and a template.

Exercise 6 Use the same methodology we used for Chip to figure out what big-bang handlers we need (to-draw + how/when the game changes + stop-when if the game stops (it does)). Then use that to write your wishlist.

Exercise 7 Complete the functions in your wishlist.

Exercise 8 Define the main function, which "composes" the handlers and the rendering functions. What input makes sense here?

Exercise 9 Define 3 numeric constants, `SHRINK-TIME0`, `SHRINK-TIME1`, and `SHRINK-TIME2` such that `(< SHRINK-TIME0 SHRINK-TIME1 SHRINK-TIME2)`. These constants represent times in the game and shrink-world uses these times to determine how quickly to shrink the world. As the time increases, the balloon should shrink faster.

Exercise 10 Design the function *shrink-world*, which takes an SW. It increments the world's time and shrinks the world's size depending on how the world's time compares with `SHRINK-TIME0`, `SHRINK-TIME1`, and `SHRINK-TIME2`.

Hint: That's **two** distinct tasks. You may choose by how much the world should shrink in each interval.

Before You Go...

If you had trouble finishing any of the exercises in the lab or homework, or just feel like you're struggling with any of the class material, please feel free to come to office hours and talk to a TA or tutor for additional assistance.