Recap: Defining constants
General form for defining a constant:

\[(\text{define } \langle \text{name} \rangle \langle \text{expression} \rangle)\]
To evaluate a constant definition,

1. Evaluate the expression and record the resulting value as the value of the constant with the given name.

To evaluate a defined constant name,

1. The value is the recorded name.
Names are arbitrary

The following is silly, but legal:

Welcome to DrRacket.
> (define FIVE 6)
> FIVE
6
> (define SIX 5)
> SIX
5
Several constants may have the same value:

Welcome to DrRacket.
> (define SEVEN 7)
> SEVEN
7
> (define SEPT 7)
> SEPT
7
If we define constants

\[
\begin{align*}
& (\text{define WIDTH 400}) \\
& (\text{define HEIGHT 600})
\end{align*}
\]

Now if we write

\[
(*) \text{ WIDTH HEIGHT}
\]

it gets evaluated:

\[
\begin{align*}
\rightarrow & \ ((*) \ 400 \ \text{HEIGHT}) \\
\rightarrow & \ ((*) \ 400 \ 600) \\
\rightarrow & \ 240000
\end{align*}
\]
Working with atomic data
Functions for images
(require 2htdp/image)

(define WIDTH 400)
(define HEIGHT 600)

(define CAT)

To add an image, copy it, e.g., this cat is copied from HtDP, §1.4
> (rotate -10 CAT)

> (rotate 10 CAT)
(define **RCAT** (rotate -10 CAT))
(define **LCAT** (rotate 10 CAT))
Functions for numbers
> (sqrt 16)
> (sqrt 16)
4
> (sqrt 16)
4
> (sqrt 11)
> (sqrt 16)
4
> (sqrt 11)
#i3.3166247903554
> (sqrt 16)
4
> (sqrt 11)
#i3.3166247903554
> (sqrt 2)
> (sqrt 16)
4
> (sqrt 11)
#i3.3166247903554
> (sqrt 2)
#i1.4142135623730951
\begin{verbatim}
> (sqrt 16)
4
> (sqrt 11)
#i3.3166247903554
> (sqrt 2)
#i1.4142135623730951
> (inexact->exact (sqrt 2))
\end{verbatim}
> (sqrt 16)
4

> (sqrt 11)
#i3.3166247903554

> (sqrt 2)
#i1.4142135623730951

> (inexact->exact (sqrt 2))
1.4142135623730951
> (sqrt 16)
> (sqrt 16)
4
> (sqrt 16)
4
> (expt 2 4)
\[
\begin{align*}
\texttt{> (sqrt 16)} & \quad \texttt{4} \\
\texttt{> (expt 2 4)} & \quad \texttt{16}
\end{align*}
\]
> (sqrt 16)
4
> (expt 2 4)
16
> (sqrt (expt 2 4))
> (sqrt 16)
4
> (expt 2 4)
16
> (sqrt (expt 2 4))
4
> (sqrt 16)
4
> (expt 2 4)
16
> (sqrt (expt 2 4))
4

Parentheses because it’s a function call
> (sqrt 16)
4
> (expt 2 4)
16
> (sqrt (expt 2 4))
4
> (sqrt (define SIXTEEN 16))

What should this be?
> (/ 10 3)
3.3

> (quotient 10 3)
3

> (modulo 10 3)
1
String functions
> (make-string 5 #\e)
> (make-string 5 \e)
"eeeee"
> (make-string 5 #\e)  
"eeeee"  

This is the character ‘e’
This is the string “eeeee”
> (make-string 5 #\e)  This is the character ‘e’
"eeeee"
> (string-length "hello")  This is the string “eeeee”
> (make-string 5 #\e)  This is the character ‘e’
"eeeeee"          This is the string “eeeeee”
> (string-length "hello")
5
> (make-string 5 #\e)  This is the character ‘e’
"eeeee"  This is the string “eeeee”
> (string-length "hello")
5
> (string-length (make-string 5 #\z))
> (make-string 5 #\e)   This is the character ‘e’
"eeeee"               This is the string “eeeee"
> (string-length "hello")
5
> (string-length (make-string 5 #\z))
5
> (make-string 5 #\e)  \textit{This is the character} \textit{‘e’}
"eeeee"  \textit{This is the string} \textit{“eeeee”}
> (string-length "hello")
5
> (string-length (make-string 5 #\z))
5
> (string-ref "Vassar" 2)
> (make-string 5 #\e)  This is the character ‘e’
"eeeeee"  This is the string “eeeeee”
> (string-length "hello")
5
> (string-length (make-string 5 #\z))
5
> (string-ref "Vassar" 2)
#\s
> (make-string 5 #\e)  
This is the character ‘e’
"eeeee"
> (string-length "hello")
5
> (string-length (make-string 5 #\z))
5
> (string-ref "Vassar" 2)
#\s
Defining functions
Remember functions from middle-school math:

Given $f(x) = 2 \cdot x$

$f(2) = 2 \cdot 2 = 4$

$f(6) = 2 \cdot 6 = 12$
To form a function definition:

\[
(\text{define} \ (\langle \text{function-name} \rangle \ \langle \text{arg-name} \rangle \ ...) \ \langle \text{expression} \rangle)
\]
(define (add-three num)
  (+ num 3))
(define (add-three num)
  (+ num 3))

special word to remember something
(define (add-three num)
  (+ num 3))
(define (add-three num)
  (+ num 3))
(define (add-three num)
  (+ num 3))
(define (add-three num)
  (+ num 3))
(define (add-three num)
  (+ num 3))
(define (add-three num)
  (+ num 3))

transform the data
Evaluation is the same as in algebra

\[
\text{(define (f x) (+ (cos x) 2))}
\]

\[
(f 0)
\rightarrow (+ (cos 0) 2)
\rightarrow (+ 1 2)
\rightarrow 3
\]
Aside: Using the Stepper
(require 2htdp/image)

;;; Draw a traffic light
(above (circle 40 "solid" "red")
  (circle 40 "solid" "yellow")
  (circle 40 "solid" "green"))
(require 2htdp/image)

;; Draw a traffic light
(above (circle 40 "solid" "red")
  (circle 40 "solid" "yellow")
  (circle 40 "solid" "green"))
(require 2htdp/image)

;; Draw a traffic light
(above
  (circle 40 "solid" "red")
  (circle 40 "solid" "yellow")
  (circle 40 "solid" "green"))

Unchanging Varying
(require 2htdp/image)

;; Draw a traffic light
(above (circle 40 "solid" "red")
   (circle 40 "solid" "yellow")
   (circle 40 "solid" "green"))

;; Can be changed to
(define (bulb color)
   (circle 40 "solid" color))

(above (bulb "red")
   (bulb "yellow")
   (bulb "green"))
We started by talking about computation, e.g., how do we interpret the expression

\((+ \ 1 \ (* \ 2 \ 3))\)
We started by talking about *computation*, e.g., how do we interpret the expression

\[(+ \ 1 \ (\times \ 2 \ 3)) \rightarrow (+ \ 1 \ 6)\]
We started by talking about *computation*, e.g., how do we interpret the expression

\[
(+ 1 (* 2 3))
\]

→ \((+ 1 6)\)

→ 7
We started by talking about *computation*, e.g., how do we interpret the expression

\[(+ 1 (* 2 3))\]

\[\rightarrow (+ 1 6)\]

\[\rightarrow 7\]

But as we start writing function definitions, we need to think about *programming*. 
Function signatures and purpose statements
;; add-three : Number -> Number
;; Add three to the given number
(define (add-three x)
  (+ x 3))
(define (sqre dimen)
  (rectangle dimen dimen "solid" "blue"))
;; sqre : Natural -> Image
;; Make a solid blue square of the given side length
(define (sqre dimen)
  (rectangle dimen dimen "solid" "blue"))
(define (mad-math a b c)
  (* (/ a b) (+ c c)))
;; mad-math : Number, NonNegNumber, Number -> Number
;; Compute \( a \div b \cdot 2c \)
(define (mad-math a b c)
  (* (/ a b) (+ c c)))
(define (rectangle-area r)
  (* (image-height r) (image-width r)))

;;; rectangle-area : Image -> Natural
;;; Return the rectangular area of the image
(define (rectangle-area r)
  (* (image-height r) (image-width r)))
(define (dupe-string str)
  (string-append str str))
;; dupe-string : String -> String
;; Return a string containing the input repeated
(define (dupe-string str)
  (string-append str str))
Acknowledgments

This lecture incorporates material from:

Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Gregor Kiczales
Shriram Krishnamurthi
Marc Smith
Laney Strange