Booleans and Conditionals

10 September 2020
Where are we?
In Racket, we can work with a variety of data types, including

- numbers
- strings (and characters)
- images
When we want to re-use a value or give it a descriptive name, we can define a constant:

(define MONTH "September")
When we want to be able to repeat a computation for different data, we can define a function, e.g.,

```
(define (greet name)
  (string-append "Hello " name "!"))
```
Testing code & why we care
Eligible for Free Shipping
- Free Shipping by Amazon
All customers get FREE Shipping on orders over $25 shipped by Amazon

Kindle Unlimited
- Kindle Unlimited Eligible

Department
- Comedy Department
- Books
- Literature & Fiction
- Romance

Avg. Customer Review
- 4.8 out of 5 stars
- 274 customer reviews

Book Series
- Penguin Classics
- The Coarse
- The Isle of Destiny Series
- Pemberton Estates
- Kit's the Wallower
- Regency Love
- The Rules of Love
- The Smoky Hills Academy
- Ladies of Bath
- The Hardwood Collection
- The Jane Austen Collection

Deals
- Today's Deals
- New Releases
- Last 30 days
- Last 90 days
- Coming soon

Book Format
- Paperback
- Hardcover
- Kindle Edition
- Large Print
- Audible Audiobook
- Audio CD
- Board Book

Author
- Jane Austen

The Jane Austen Collection: Deluxe 6-Volume Slipcase Edition (Arcturus Collector's Classics)
by Jane Austen  |  Sep 15, 2016

Hardcover
159 pages

Prime Get it as soon as Wed, Sep 9
FREE Shipping by Amazon

More Buying Choices
$17.67 (12 used & new offers)

Kindle
105 Kindle Editions
Available Instantly

Paperback

Other format: Audio CD

The Jane Austen Society: A Novel
by Natalie Jenner  |  May 28, 2020

Hardcover

Get it as soon as Wed, Sep 9
FREE Shipping on your first order shipped by Amazon

More Buying Choices
$15.68 (44 used & new offers)

Kindle
11 Kindle Editions
Available Instantly

Audible Audiobook
102 Audible Editions
Free with Audible trial

Other formats: Audio CD, Paperback

Jane Was Here: An Illustrated Guide to Jane Austen's England
by Nicole A. Jacobson, Devon MacLaren, et al.  |  Jan 8, 2020

Hardcover
159 pages

Get it as soon as Wed, Sep 9
FREE Shipping on orders over $25 shipped by Amazon.
The Jane Austen Collection: Deluxe 6-Volume Slipcase Edition (Arcturus Collector's Classics)
by Jane Austen \ | Sep 15, 2016

Hardcover

$24

Kindle

$15

Paperback

$17.60

The Jane Austen Society: A Novel
by Natalie Jenner \ | May 26, 2020

Hardcover

$16

Kindle

$15

Audible Audiobook

$15

Jane Was Here: An Illustrated Guide to Jane Austen's England
by Nicole Jacobson, Devon McLean, et al. \ | Jun 3, 2020

Hardcover

$15
;; add-three : Number -> Number
;;    Add three to the given number
(define (add-three x)
  (+ x 3))
;; add-three : Number -> Number
;;   Add three to the given number

(check-expect (add-three 0) 3)
(check-expect (add-three 107.5) 110.5)
(check-expect (add-three -15) -12)

(define (add-three x)
  (+ x 3))
[See example code.]
Booleans and if expressions
#true
#false

Can be abbreviated to #t and #f
> (and #true #false)
#false
> (or #true #false)
#true
> (and (< 1 2) (> 2 3))
#false
> (or (<= 1 0) (= 1 1))
#false
To combine Boolean values, we can use **and**: 

\[(\text{and } \langle \text{expr1} \rangle \langle \text{expr2} \rangle \ldots )\]

and **or**: 

\[(\text{or } \langle \text{expr1} \rangle \langle \text{expr2} \rangle \ldots )\]

Evaluation of **and** stops – is “short-circuited” – as soon as an expression evaluates to false.

Evaluation of **or** stops as soon as an expression evaluates to true.
To change an expression that evaluates to #true to be #false or vice versa, use not.

> (not (= 1 0))
#true
(require 2htdp/image)

(define I1 (rectangle 10 20 "solid" "red"))
(define I2 (rectangle 20 10 "solid" "blue"))

(<= (image-width I1) (image-width I2))
(require 2htdp/image)

(define rect (rectangle 10 20 "solid" "red"))

(if (< (image-width rect) (image-height rect))
  "tall"
  "wide")
To form an **if** expression:

```
(if ⟨expression⟩
   ⟨expression⟩
   ⟨expression⟩)
```

- **True–false question**
- **True (“then”) answer**
- **False (“else”) answer**
Evaluation rule for *if* expressions

1. If the question expression is not a value, evaluate it, and replace with value

2. If the question is true, replace entire *if* expression with true answer expression

3. If the question is false, replace entire *if* expression with false answer expression

4. The question is a value other than true or false, so produce an error
Booleans and cond expressions
What if, instead of producing a Boolean to say if an image is tall or not, we classify them as “tall”, “square”, or “wide”?
(require 2htdp/image)

(define rect (rectangle 10 20 "solid" "red"))

(if (< (image-width rect) (image-height rect))
   "tall"
   (if (= (image-width rect) (image-height rect))
       "square"
       "wide")))
(require 2htdp/image)

(define rect (rectangle 10 20 "solid" "red"))

(cond [(< (image-width rect) (image-height rect))
   "tall"
   [ (= (image-width rect) (image-height rect))
     "square"
   [else "wide"]])

Ah, better! 😌
Evaluation rules for cond

If the first question is \texttt{false}, throw away the first line.

For example,

\[
(+ 1 (\texttt{cond \#false 1} \\
\texttt{\#true 17})))
\]
\[
→ (+ 1 (\texttt{cond \#true 17})))
\]
\[
→ (+ 1 17)
\]
\[
→ 18
\]
Evaluation rules for \texttt{cond}

If the first question isn’t a value yet, evaluate the first question as a subexpression.

For example,

\[
(+ \ 1 \ (\texttt{cond} \ \[\ (< \ 1 \ 2) \ 5]\ \ [\texttt{else} \ 8]))
\rightarrow (+ \ 1 \ (\texttt{cond} \ \[\#true \ 5]\ \ [\texttt{else} \ 8]))
\rightarrow (+ \ 1 \ 5)
\rightarrow 6
\]
Evaluation rules for `cond`

If there are no true answers, produce an error.

For example,

```
(cond)
→ 😞 Error!
```
Predicates and combining Booleans
Primitives or other functions that produce a Boolean value are called *predicates*, e.g., for numbers,

```plaintext
> (> 100 100)  #false
> (= 100 100)   #true
```
What if the operands aren’t numbers?
> (= "hello" "goodbye")
> (= "hello" "goodbye")

> (= \a \a)
> (= "hello" "goodbye")

> (= #\a #\a)

> (= #true #false)
> (= "hello" "goodbye")

> (= \a \a)

> (= #true #false)

> (= (circle 10 "solid" "red")
   (circle 10 "solid" "red"))
> (=  "hello"  "goodbye")
=: expects a number as 1st argument, given  "hello"

> (=  #\a  #\a)
=: expects a number as 1st argument, given  #\a

> (=  #true  #false)
=: expects a number as 1st argument, given  #true

> (=  (circle 10  "solid"  "red")
   (circle 10  "solid"  "red"))
=: expects a number as 1st argument, given  #<image>
> (= "hello" "goodbye")
> (string=? "hello" "goodbye")
> (= "hello" "goodbye")
> (string=? "hello" "goodbye")

> (= \a \a)
> (char=? \a \a)
> (= "hello" "goodbye")
> (string=? "hello" "goodbye")

> (= \a \a)
> (char=? \a \a)

> (= #true #false)
> (boolean=? #true #false)
> (= "hello" "goodbye")
> (string=? "hello" "goodbye")

> (= \a \a)
> (char=? \a \a)

> (= #true #false)
> (boolean=? #true #false)

> (= (circle 10 "solid" "red")
   (circle 10 "solid" "red"))
> (image=? (circle 10 "solid" "red")
          (circle 10 "solid" "red"))
Are You My Data Type?

by P.D. Eastman
> (number? 18)
> (number? 18)
#true
> (number? 18)

#true

Calling a predicate function
> (number? 18)  
true

- Calling a predicate function
- Boolean literal
> (number? 18)

#true
> (number? 18)
#true
> (number? "18")
> (number? 18)
#true
> (number? "18")
#false
> (number? 18)
#true
> (number? "18")
#false
> (define BOX
  (rectangle 20 80 "solid" "brown"))
> (number? 18)
#true
> (number? "18")
#false
> (define BOX
    (rectangle 20 80 "solid" "brown")
> (number? BOX)
> (number? 18)
#true
> (number? "18")
#false
> (define BOX
   (rectangle 20 80 "solid" "brown")
> (number? BOX)
#false
> (number? 18)
#true
> (number? "18")
#false
> (define BOX
   (rectangle 20 80 "solid" "brown")
> (number? BOX)
#false
> (image? BOX)
> (number? 18)
#true
> (number? "18")
#false
> (define BOX (rectangle 20 80 "solid" "brown")
> (number? BOX)
#false
> (image? BOX)
#true
Functions with branches
Maybe you watched Game of Thrones.
How does the season/episode you’re watching affect you?
hours-till-bed: Number, Number -> Number
Calculate the number of hours till you can sleep based on your current place in GoT season
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep))
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))

If it’s the season finale, two hours till bed!
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))

If it’s the season finale, two hours till bed!

Otherwise, one hour till bed
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))

A Boolean value can come from

Calling a predicate function

A literal or constant

A comparison
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))

If the Boolean is #true

A Boolean value can come from

Calling a predicate function

A literal or constant

A comparison
Let's define a function to calculate the number of hours till you can sleep based on your current place in GoT season.

```scheme
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))
```

This function takes two arguments: the season and the episode number. It uses an if statement to determine the number of hours till bedtime:

- If the season is a finale, it returns 2.
- Otherwise, it returns 1.

A Boolean value can come from:

- Calling a predicate function
- A literal or constant
- A comparison
;; hours-till-bed : Number, Number -> Number
;; Calculate the number of hours till you can sleep based
;; on your current place in GoT season
(define (hours-till-bed season ep)
  (if (is-finale? season ep) 2 1))

This is a predicate function, but we haven’t defined it yet!
Let’s write the `is-finale?` function.
> (is-finale? 7 5)
> (is-finale? 7 5)

Predicate function we want to call
> (is-finale? 7 5)

Season number
> (is-finale? 7 5)

Episode number
is-finales? : Number, Number -> Boolean
Determine whether the given season/episode numbers were a season finale
;; is-finale? : Number, Number -> Boolean
;;   Determine whether the given season/episode numbers were
;;   a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)
;; is-finale? : Number, Number -> Boolean
;; Determine whether the given season/episode numbers were
;; a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)

(define (is-finale? season ep) ???)
;; is-finale? : Number, Number -> Boolean
;;   Determine whether the given season/episode numbers were
;;   a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)

(define (is-finale? season ep)
  (cond [ ]))
;; is-finale? : Number, Number -> Boolean
;;   Determine whether the given season/episode numbers were
;;   a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)

(define (is-finale? season ep)
  (cond [ (< season 7) (= ep 10) ]))
;; is-finale? : Number, Number -> Boolean
;;    Determine whether the given season/episode numbers were
;;    a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)

(define (is-finale? season ep)
  (cond [(and (< season 7) (= ep 10)) ]))
;;;; is-finale? : Number, Number -> Boolean
;;;; Determine whether the given season/episode numbers were
;;;; a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)

(define (is-finale? season ep)
  (cond [(and (< season 7) (= ep 10)) #true])))
;; is-finale? : Number, Number -> Boolean
;; Determine whether the given season/episode numbers were
;; a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)

(define (is-finale? season ep)
  (cond [(and (< season 7) (= ep 10)) #true]
        [(and (= season 7) (= ep 7)) #true])))
define (is-finaled? season ep)  
  (cond [(and (< season 7) (= ep 10)) #true]  
      [(and (= season 7) (= ep 7)) #true]  
      [(and (= season 8) (= ep 6)) #true]]

(check-expect (is-finaled? 6 10) #true)  
(check-expect (is-finaled? 6 9) #false)  
(check-expect (is-finaled? 7 8) #true)  
(check-expect (is-finaled? 7 7) #false)  
(check-expect (is-finaled? 8 6) #true)
;; is-finale? : Number, Number -> Boolean
;;    Determine whether the given season/episode numbers were
;;    a season finale

(check-expect (is-finale? 6 10) #true)
(check-expect (is-finale? 6 9) #false)
(check-expect (is-finale? 7 8) #true)
(check-expect (is-finale? 7 7) #false)
(check-expect (is-finale? 8 6) #true)

(define (is-finale? season ep)
  (cond [(and (< season 7) (= ep 10)) #true]
        [(and (= season 7) (= ep 7)) #true]
        [(and (= season 8) (= ep 6)) #true]
        [else #false])))
Back to *hours-till-bed*...
Acknowledgments

This lecture incorporates material from:


Gregor Kiczales, University of British Columbia

Marc Smith, Vassar College

Laney Strange, Northeastern University