Goals of this tutorial

You should be able to...

- understand and perform **Boolean algebra**.
- understand and use **conditional expressions**.
CQ: For the function `expt`, which of the following has the best purpose, contract, and example?

A

```
;; (expt arg1 arg2) produces the first
;;   number to the power of the
;;   second number
;; expt: num num → num
;; Example:
(check-expect (expt 3 2) 9)
```

B

```
;; (expt arg1 arg2) consumes two
;;   numbers, arg1 and arg2, and
;;   produces the result of the first
;;   number to the power of the second
;;   number
;; (expt arg1 arg2): Num Num → Num
;; Example:
(check-expect 9 (expt 3 2))
```

C

```
;; Contract: expt: Number Num → Num
;; Purpose: (expt x y) produces arg1
;;   to the power of arg2
;; Example:
(check-expect (expt 4 2) 16)
```

D

```
;; (expt arg1 arg2) produces arg1 to
;;   the power of arg2
;; expt: Num Num → Num
;; requires: if arg1 is not zero, then the
;;   produced number is also
;;   not zero
(check-expect 8 (expt 2 3))
```

E

```
;; (expt arg1 arg2) produces arg1 to
;;   the power of arg2
;; expt: Num Num → Num
;; Example:
(check-expect (expt 3 2) 9)
```

Note that `expt` is a built-in Racket function.
Review: Boolean-valued functions

Boolean-valued functions produce Boolean values: true and false. These functions are also called predicates.

Standard Racket uses #t and #f, or #true and #false; these will sometimes show up in basic tests and correctness tests.

Racket provides many built-in Boolean functions (for example, to do numerical comparisons: \((\geq x y), (\leq x y)\)).
Review: Boolean-valued functions

Note that comparison functions are often specific to certain data types (for example, $(= a b)$ vs. $(\text{symbol}=? x y)$, where $a$ and $b$ are numbers, but $x$ and $y$ are symbols).

The naming convention for most predicates and Boolean parameters is to append a question mark to the name (for example, $\text{even}?$, $\text{symbol}?$, $\text{expired}?$).
Review: Boolean Operators

`and` and `or` are special forms in Racket.

`and` and `or` may have two or more arguments.

Their arguments are evaluated from left to right.

`and`:

- If an argument evaluates to `false`, the entire expression evaluates to `false`.
- Otherwise, the next argument is evaluated.
- If there are no arguments remaining, the expression evaluates to `true`.


Review: Boolean Operators

or:

- If an argument evaluates to true, the entire expression evaluates to true.
- Otherwise, the next argument is evaluated.
- If there are no arguments remaining, the expression evaluates to false.

not:

- not must have exactly one argument.
- If the argument evaluates to true, the entire expression evaluates to false.
- If the argument evaluates to false, the entire expression evaluates to true.
Clicker Question - Boolean Expressions

Which of the following expressions evaluates to true?

A \((\text{'IAMSMART 'IAMSMART})\)

B \((\text{not (not false)})\)

C \((\text{check-expect (max 10 \((\times 3 (\text{min (\text{+ 3 4) 8)))}) 21)})\)

D \((\text{or \((\text{= 27 (expt 3 3)) (\text{< (sqr 5) 28}) (\times (sqr 10) 5)})\)})\)

E \((\text{and true (not true) (not (not (true)))})\)
Group Problem - valid-pin?

The Iron Bank of Braavos has the following rules for setting a Bank PIN:

- A Bank PIN must be a 4-digit positive integer.
- The last two digits of a Bank PIN must be a multiple of 7, but cannot be divisible by 6 or 9.

For example, 1049 is considered a valid Bank PIN, but 9999 is not. Using only boolean expressions, first write a helper function last-two-digits that consumes a number n and produces the last two digits of n. Include a contract.
Group Problem - valid-pin?

The Iron Bank of Braavos has the following rules for setting a Bank PIN:

- A Bank PIN must be a 4-digit positive integer.
- The last two digits of a Bank PIN must be a multiple of 7, but cannot be divisible by 6 or 9.

Next, write the function `valid-pin?` that consumes a number, and produces `true` if the number is considered a valid PIN according to the rules above, and `false` otherwise. Include a purpose, contract, and examples.
Review: Conditional Expressions

The general form of a conditional expression is

```
(cond
  [question1 answer1]
  [question2 answer2]
  …
  [questionk answerk])
```

where questionk could be else.
• Each of the questions must evaluate to a **boolean** value.

• The questions are evaluated from **top to bottom**.

• If a question evaluates to **true**, no more questions are evaluated and the cond expression is reduced to just the answer for that question.

• If none of the questions evaluate to **true**, then the result is the answer in the **else** clause.

• If there are no questions that evaluate to **true** and there is no **else** clause, then Racket will report an error.
Clicker Question - Cond Expression

What does the following cond expression evaluate to?

```
(cond
    [(< 180 180) 'yellow]
    [(and (not false) (= (max 4 -6) (sqr 2))) 'red]
    [(= (/ 3 (sqrt 9)) 1) 'yellow]
    [else 'green])
```

A  'blue  
B  'red  
C  'yellow  
D  'green  
E  Nothing. There is an error.
Group Problem - Converting cond to booleans

Rewrite the function \( f \) without \texttt{cond}. You may use \texttt{and}, \texttt{or} and \texttt{not}.

\[
\begin{align*}
&\text{(define } (f \ x) \\
&\quad \text{(cond} \\
&\quad \quad [(\text{not } (\text{p1? } x)) \ (\text{p2? } x)] \\
&\quad \quad [\text{else } (\text{p1? } x)])}
\end{align*}
\]
Shortening cond

Do not compare boolean variables with true/false as they can be used directly as a condition. The examples below have the exact same behaviour:

```
(define (f x)
  (cond
    [(boolean=? (p1? x) true) (f1 x)]
    [else (f2 x)]))
```

```
(define (f x)
  (cond
    [(p1? x) (f1 x)]
    [else (f2 x)]))
```
Shortening cond

Nested cond expression can be shortened: the following are equivalent:

```scheme
(define (f x)
    (cond
        [(p1? x) (cond
            [(p2? x) (f1 x)]
            [else (f2 x)])]
        [else (f3 x)])
)
```

```scheme
(define (g x)
    (cond
        [(and (p1? x) (p2? x)) (f1 x)]
        [(p1? x) (f2 x)]
        [else (f3 x)])
)
```
Note: Tests for conditional expressions

- Test for each clause in the cond expression

- If the function specifications allow for this:
  - Test all boundary points
  - Write at least one test for each interval (not including the boundary)

- DrRacket highlights unused code
  - Having no code highlighted does not mean that your code is fully tested
  - However, highlighted code means your testing is incomplete
Clicker Question - Testing

(define (foo x)
  (cond
    [(< x 0) (exp x)]
    [(<= x 100) (sub1 x)]
    [(< x 1000) x]
    [else (add1 x)])
)

Minimally, how many tests would be required for this function?

A  4
B  5
C  6
D  7
E  8
Group Problem - receives-discount/bool?

A warehouse store discounts its merchandise according to the following rules:

- All items in the clearance section are discounted.
- If an item has been in the store for at least 6 weeks, it is only discounted if the item is an ‘appliance or ‘clothing.
- If an item has been in the store for at least 3 weeks, but less than 6 weeks, it is only discounted if the item is a ‘food.
- All other items are not discounted.

Using only Boolean operations (and, or, or not), write a function receives-discount/bool? that consumes the number of weeks an item has been in the store, a symbol representing the type of the item, and a Boolean value representing whether the item is in the clearance section. The function produces true if the item receives a discount, and false otherwise. You only need to write the function definition.
Group Problem - receives-discount/bool?

(define (receive-discount/bool? weeks type clearance?)
  (or clearance?
      (and (>= weeks 6) (or (symbol=? type 'appliance)
                            (symbol=? type 'clothing)))
      (and (>= weeks 3) (< weeks 6) (symbol=? type 'food))))

• Note that the clearance? parameter was used directly as an argument to or.

• When working with boolean parameters, there should be no need to use equality predicates (such as equal? or boolean=?) to determine if a parameter is true or false.