# CS 135 Fall 2015

# Tutorial 1



## Goals of this tutorial

You should be able to...

- write the full **design recipe** for simple arithmetic functions, functions involving Boolean data, and functions using conditional expressions.
- understand and perform **Boolean algebra**.
- understand and use **conditional expressions**.



## Review: The five design recipe components

**Purpose:** Describes what the function is to produce. You should include parameter names in your purpose statement.

**Contract:** Describes what type of arguments the function consumes and what type of value it produces.

**Additional contract requirements:** If there are important constraints on the parameters that are not fully described in the contract, add an additional **requires** section to "extend" the contract.

**Examples:** Illustrating the use of the function.

**Definition:** The Racket definition (header and body) of the function.

**Tests:** A thorough set of inputs and expected outputs.

# Clicker Question - Design Recipe

For the built-in function expt, which of the following is the best design recipe (ignoring tests)?

**A**

```
;; (expt arg1 arg2) raises arg1 to
;;     the power of arg2
;; expt: num num ⟶ num
;; Example:
(check-expect (expt 3 2) 9)
(define (expt arg1 arg2) ...)
```

**B**

```
;; (expt arg1 arg2) raises arg1 to
;;     the power of arg2
;; expt: Num Num ⟶ Num
;; Example:
(check-expect (expt 3 2) 9)
(define (expt arg1 arg2) ...)
```

**C**

```
;; expt: Nat Nat ⟶ Nat
;; Purpose: To raise a natural number to a power
;; Example:
(check-expect (expt 4 2) 16)
(define (expt arg1 arg2) ...)
```

**D**

```
;; (expt arg1 arg2) raises arg1 to
;;     the power of arg2
;; expt: num num ⟶ num
;; Example:
(= (expt 2 3) 8)
(define (expt arg1 arg2) ...)
```

**E**

```
;; (expt arg1 arg2) raises an integer to a power
;; expt: Int Int ⟶ Int
;; Example:
(check-expect (expt 3 2) 9)
(define (expt arg1 arg2) ...)
```



# How to find the help pages

- DO NOT use Google search. It will land you at the wrong language level, typically at the full Racket help page.
- Open DrRacket: Help menu $>$ Help Desk (this opens a browser window) $>$ Teaching $>$ How to Design Programs Languages $>$ Select the appropriate language level (e.g. Beginning Student).
- NOTE the categorized list of functions on the left side bar.
- If you must Google, then you have to add the teaching language name to your query, e.g. ”racket beginning student”.



# Purpose & Contract

Using the help desk create a purpose and contract for even?.

## Purpose & Contract

Using the help desk create a purpose and contract for modulo.



## Group Problem - Tasty Beverages

You occasionally host parties for your friends at school. You always provide a box of 20 tasty beverages for your guests, and you always count how many guests you have. But you'd like to know how many of your friends from CS 135 came. You notice that each of your CS 135 classmates consumes two beverages at the party, while everyone else just has one.

Create a Racket function tastyb with a complete design recipe that consumes the number of tasty beverages you end with and the number of people at the party. The function will produce the number of CS 135 classmates at the party.



## Review: Boolean-valued functions

A function which tests whether two numbers $x$ and $y$ are equal has two possible Boolean values: true and false.

Racket provides many built-in Boolean functions (for example, to do comparisons: (= x y), (>= x y)).

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

## Review: Complex Relationships

You may have learned in Math 135 how propositions can be combined using the connectives AND, OR, NOT. Racket provides the corresponding functions and, or, not. These are used to test complex relationships.

The functions and, or may have more than two arguments.

The function and has value true exactly when all of its arguments have value true.

The function or has value true exactly when at least one of its arguments has value true.

The function not has value true exactly when its one argument has value false.



## Clicker Question - Boolean Expression

Which of the following expressions evaluates to true?

**A** (= 'blue 'blue)
**B** (not (not false))
**C** (check-expect (+ 3 7) 10)
**D** (or (= 25 24) (< 27 28))
**E** (or false (not true))



## Group Problem - pair?

Write a Racket function, pair? with full design recipe which consumes 4 values of a standard playing card (Natural numbers from 1-13). And returns true if any of the two cards are the same and false otherwise. For this question, you may only use Boolean expressions (no cond allowed).

## Review: Conditional Expressions

The general form of a conditional expression is

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

where questionk could be else.



The questions are evaluated in order; as soon as one evaluates to true, the corresponding answer is evaluated and becomes the value of the whole expression.

- The questions are evaluated in top-to-bottom order.
- As soon as one question is found that evaluates to true, no further questions are evaluated.
- Only one answer is ever evaluated.

  (the one associated with the first question that evaluates to true, or associated with the else if that is present and reached)



## Group Problem - flush?

In many card games the term flush is used to denote when an entire hand has the same suit. Given the suits (as symbols) of a four card hand, write a function flush? that returns true if the hand is a flush and false otherwise. For this problem, try to use only cond statements (no 'and', 'or', or 'not'). Include a design recipe (you may only write one check-expect expression for each of Example and Test).

## Group Problem - prime? Boolean

Write a function prime? that consumes an integer between 2 and 120 (inclusive) and returns true if that number is prime. For this question, you may only use Boolean expressions (no cond). Include a design recipe (you may only write one check-expect expression for each of Example and Test).



## Group Problem - prime? Conditional

Rewrite an implementation of prime? that only uses cond statements (no booleans allowed). You don't need to include design recipe.