Don’t program defensively. Use contracts instead.

A *contract* describes the behavior of functions and methods in terms of code, often just boolean assertions or function “types” with boolean assertions. Contracts supplement conventional type systems and can also be used on their own.

conventional programming

```
#; {type Results   = List Rankings [Listof Player]]}
;; the second list represents all those players that cheated or failed

#; {type Rankings = [Listof [Listof Player]]}
;; Rankings represents all players ranked in first place, second place, etc.

(provide
 #; {Natural Natural Natural [Listof Player] -> Results}
 referee)
```

interface for clients

```
;; -----------------------------------------------------------------------------
;; [Listof Player]
;; cheaters and failing players for a run of referee
(define *failures '[])

(define (referee row col fish# players)
```

```
  (unless (and (integer? row)
               (integer-in MIN-ROW row MAX-ROW))
    (error 'referee "wrong number of rows demanded: ~a" row))

  (unless (and (integer? col)
               (integer-in MIN-COLUMN col MAX-COLUMN))
    (error 'referee "wrong number of columnss demanded: ~a" col))

  (unless (and (integer? fish#)
               (integer-in MIN-FISH fish# MAX-FISH))
    (error 'referee "wrong number of fishs demanded: ~a" fish#))

  (unless (integer-in MIN-PLAYERS (length players) MAX-PLAYERS)
    (error 'referee "wrong number of fishs demanded: ~a" fish#))
```

defensive programming

```
  (inform players 'starting)
  (start-game-with-placements players)
  (define rankings (run-movement players))
  (inform players 'game-over)
  (list (map (λ (r) (remove* *failures r)) rankings) *failures))
```

actual computation

Problem: defensive programming obscures the actual purpose of the function. In many cases, the defensive code is even co-mingled with the proper functionality —- making the problem even worse.

HtDP checked functions design

```
#; {type Results   = [List Rankings [Listof Player]]}
;; the second list represents all those players that cheated or failed

#; {type Rankings = [Listof [Listof Player]]}
;; Rankings represents all players ranked in first place, second place, etc.

(provide
 #; {Natural Natural Natural [Listof Player] -> Results}
 referee)
```

interface for clients

```
;; ---------------------------------------------------------------------------

(define (referee row col fish# players)
  (referee-contract row col fish# players)
```

check first, then compute

```
  (inform players 'starting)
  (start-game-with-placements players)
  (define rankings (run-movement players))
  (inform players 'game-over)
  (list (map (λ (r) (remove* *failures r)) rankings) *failures))
```

actual computation

```
(define (referee-contract row col fish# players)

  (unless (and (integer? row)
               (integer-in MIN-ROW row MAX-ROW))
    (error 'referee "wrong number of rows demanded: ~a" row))

  (unless (and (integer? col)
               (integer-in MIN-COLUMN col MAX-COLUMN))
    (error 'referee "wrong number of columnss demanded: ~a" col))

  (unless (and (integer? fish#)
               (integer-in MIN-FISH fish# MAX-FISH))
    (error 'referee "wrong number of fishs demanded: ~a" fish#))

  (unless (integer-in MIN-PLAYERS (length players) MAX-PLAYERS)
    (error 'referee "wrong number of fishs demanded: ~a" fish#)))
```

separate checker

Problem: the author of a client module must still read the internals of the module to figure out the checking and potential errors.

program design with contracts

```
(define ranking/c [listof [listof player/c]])
;; Rankings represents all players ranked in first place, second place, etc.

(define results/c [list/c ranking/c [listof player/c]])
;; the second list represents all those players that cheated or failed

(provide
 (contract-out
  [referee
   (-> (integer-in MIN-ROW MAX-ROW)
       (integer-in MIN-COLUMN MAX-COLUMN)
       (integer-in MIN-FISH MAX-FISH)
       (compose (integer-in MIN-PLAYERS MAX-PLAYERS) length)
       results/c)]))
```

interface for clients who no longer need to read any module code

```
;; ---------------------------------------------------------------------------
;; [Listof Player]
;; cheaters and failing players for a run of referee
(define *failures '[])
```

```
(define (referee row col fish# players)
  (inform players 'starting)
  (start-game-with-placements players)
  (define rankings (run-movement players))
  (inform players 'game-over)
  (list (map (λ (r) (remove* *failures r)) rankings) *failures))
```

actual computation

Another advantage: once programmers use contracts, it is easy to strengthen the contract with additional elements to protect against accidental misuses and catch them early.

program design with contracts

```
(define ranking/c [listof [listof player/c]])
;; Rankings represents all players ranked in first place, second place, etc.

(define results/c [list/c ranking/c [listof player/c]])
;; the second list represents all those players that cheated or failed

(provide
 (contract-out
  [referee
   (->i
    ([row (integer-in MIN-ROW MAX-ROW)]
     [col (integer-in MIN-COLUMN MAX-COLUMN)]
     [fish (integer-in MIN-FISH MAX-FISH)]
     [plrs (compose (integer-in MIN-PLAYERS MAX-PLAYERS) length)])
    #:pre/name (row col plrs) "enough spots" (<= (avatar# plrs) (* row col))
    (r results/c))]))
;; ---------------------------------------------------------------------------
;; [Listof Player]
;; cheaters and failing players for a run of referee
(define *failures '[])

(define (referee row col fish# players)
  (inform players 'starting)
  (start-game-with-placements players)
  (define rankings (run-movement players))
  (inform players 'game-over)
  (list (map (λ (r) (remove* *failures r)) rankings) *failures))
```

interface for clients who no longer need to read any module code

strengthen invariant

actual computation