# Mixed Data (Ch. 7)

- Data types we've seen:
  - numbers
  - booleans
  - symbols
  - structures
- It's often necessary to write functions that can deal with different kinds of data in different ways.

# Predicates

- Each is a function that consumes *something* and returns a boolean (`true` or `false`)

`number?`

`boolean?`

`symbol?`

`struct?`

# Predicate Usage

```
(number? 13.4)
true
(number? 'hello)
false
(number? '17)
true
(number? (make-posn 3 4))
false
```

# When to use predicates

- Functions can make sure that arguments have the right data type.
- Functions can operate on different data types in different ways
  - function is a `cond` statement with one entry for each supported data type.

# Example

```
(define (add2nums x y)
  (cond
    [ (and (number? x)
           (number? y))
      (+ x y) ]

    [ else 0 ]))
```

# Another Example

```
; word->number converts a symbol
;   to a number
(define (word->number x)
  (cond
    [ (symbol=? x 'zero) 0]
    [ (symbol=? x 'one) 1]
    [ (symbol=? x 'two) 2]
    ...
    [ (symbol=? x 'nine) 9 ]))
```

# Example continued

```
(define (add x y)
  (+
   (cond
    [ (number? x) x]
    [ (symbol? x) (word->number x)])
   (cond
    [ (number? y) y]
    [ (symbol? y) (word->number y)])
   ))
```

# Better!

```
(define (val x)
   (cond
    [ (number? x) x]
    [ (symbol? x) (word->number x)]))

(define (addnew x y)
  (+ (val x) (val y)))
```

# Structure predicates

- The predicate structure? is true for any structure.

  `(structure? (make-posn 3 4))` is true

- When you define a structure, scheme creates a new predicate you can use to find out if *something* is one of those structures.

  `(posn? (make-posn 3 4))` is true

# Exercise

- Create a structure definition for
  - a circle (did this already)
  - a square
- Create a function that will draw either a square or a circle:

```
(draw-shape shape)
```

- `shape` is either a circle or a square

# Exercise test-code

```
(draw-shape
    (make-circle
       (make-posn 100 100) 50 'Red))

(draw-shape
    (make-square
       (make-posn 300 200) 80 'Black))
```