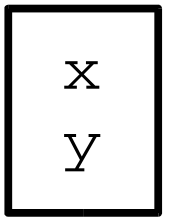
More Structures (Ch. 6)

- Scheme structure: collection of a fixed number of *named* values.
  - we've played with a `posn` structure which is defined in the draw teachpack.

```
make-posn
posn-x
posn-y
```



Defining Structures

- We can define a new *kind* (class) of structure
  - we need to specify the names of the *fields*
    - scheme will use these names to create new functions for us.
  - we need to *document* the types of the field values.
    - other programmers need to know how we *intend* each field to be used.
- Once we define a new kind of structure, scheme will create some functions for us:
  - a constructor and selectors.

`define-struct`

A new *special form*:

```
(define-struct structname (field1 field2 ...))
```

The result of evaluating `define-struct` is:

- constructor function:

```
(make-structname val1 val2 ...)
```

- selector functions:

```
(structname-field1 struct)
(structname-field2 struct)
```

`posn` structure definition

```
(define-struct posn (x y))
```

result is the functions:

```
make-posn  posn-x  posn-y
```

```
(make-posn 11 33)
(posn-x foo)
(posn-y (make-posn 1 2))
```

Some more possible structures:

A circle:

```
(define-struct circle (x y radius))
```

or possibly:

```
(define-struct circle (center radius))
```

A student:

```
(define-struct student (name age id email))
```

Data Definitions

- Scheme does not allow us to specify what the *type* of each field is.
- We should always document this so other programmers know how to use a structure.

```
(make-posn 'Hello 'There)
```

Sample Data Definitions

A `posn` is a structure:

```
(make-posn x y)
```

where `x` and `y` are numbers.

A `student` is a structure:

```
(make-student name age id email)
```

where `name` and `email` are symbols, `age` and `id` are numbers.

Functions that modify structures

- A general template: function consumes a structure and produces a structure.
  - need to call `make-`*`whatever`* inside the function
  - need to extract values from old structure to create the new one.
  - you have probably done something like this when using `posn` with drawing functions...

Example function that modifies a structure

```
(define increment-age (stu)
  (make-student
    (student-name stu)
    (+ 1 (student-age stu))
    (student-id stu)
    (student-email stu)))
```

Data Analysis and Design

- structures may seem to complicate things, but eventually you will see that they simplify.
- Large programs/systems are not possible without some level of *abstraction*
  - we like to have code that deals with "a student" rather than with a name, age, id and email
- Using structures could simplify the design of some of your drawing code.

Design Recipe

- The textbook describes changes to the proposed *design recipe* to support structures.
  - The original design recipe included the contract, purpose, examples, template, definition and test code.
  - Check out figure 12 on page 71.
- Note: The general idea of the design recipe(s) proposed in the book is essential, but you should do what makes sense to you (keep in mind that this kind of documentation is also for others.)

Exercises 6.6.1-6.6.6

- Simple animation – moves a circle across a canvas.
- Important Concept:
  - *iterative refinement:*
    - start developing a program by first writing (and testing) a simple version that deals with the most important part of the problem.
    - refine the working program so that it deals with more complex situations.
- Finish these exercises (don't stop with what you get done today!).