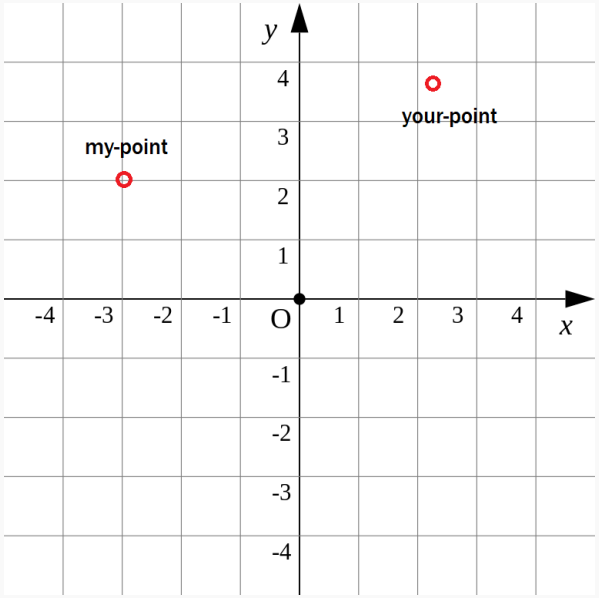
Structures

CS 151: Introduction to Computer Science I

June 22, 2018

A Tale of Two Coordinates




```
(: my-point : Point)
(: your-point : Point)

(: add-point : Point Point -> Point)
(: negate : Point -> Point)
(: x-coordinate : Point -> Real)
(: y-coordinate : Point -> Real)
```

A Tale of Two Coordinates

```
(define-struct Point
  ([x :  Real]
   [y :  Real]))
```

`define-struct` creates truly new types

```
(:  my-point :  Point)
(define my-point (Point -3 2))

(:  your-point :  Point)
(define your-point (Point 2.25 3.75))
```

Struct things you get for free

```
(define-struct Point
  ([x : Real]
   [y : Real]))
```

New type: `Point`

Constructor:

```
(: Point : Real Real -> Point)
```

Selector functions:

```
(: Point-x : Point -> Real)
(: Point-y : Point -> Real)
```

Type check function:

```
(: Point? : Any -> Boolean : Point)
```

Some more examples...

```
(define-struct 3D-Point
  ([x : Real]
   [y : Real]
   [z : Real]))
```

```
(define-struct Student
  ([name : String]
   [cnet-id : Symbol]
   [year : Symbol] ;; either 'undergrad or 'grad
   [house : String]))
```

```
(define-struct Circle
  ([radius : Real]
   [center : Point]
   [color : Symbol]))
```

Real-And-String

```
(define-struct Real-And-String
  ([real :  Real]
   [string :  String]))
```

An old function: $f(x) = x^2 + 1$

```
(:  f :  Real -> Real)
(define (f x)
  (+ (* x x) 1))
```

Also add a message to the output,

```
(:  f :  Real -> Real-And-String)
(define (f x)
  (Real-And-String
   (+ (* x x) 1)
   "Enjoy your day"))
```

Union Types: Shapes

```
(define-struct Circle
  ([radius : Real]
   [center : Point]
   [color : Symbol]))
```

```
(define-struct Square
  ([side-length : Real]
   [center : Point]
   [color : Symbol]))
```

A shape is either a Circle or a Square. Racket allows us to create a new type for shapes:

```
(U Circle Square)
```

Union Types: Shapes

What is the color of a shape?

```
(: shape-color : (U Circle Square) -> Symbol)
(define (shape-color shape)
  (cond
    [(Circle? shape) (Circle-color shape)]
    [(Square? shape) (Square-color shape)]))
```

What is the area of a shape?

```
(: area : (U Circle Square) -> Real)
(define (area shape)
  (cond
    [(Circle? shape)
     (* pi (expt (Circle-radius shape) 2))]
    [else (expt (Square-side-length shape) 2)]))
```

Type Definitions

Exactly like variable definitions!

```
(define-type Shape (U Circle Square))
```

```
(: area : Shape -> Real)
(define (area shape)
  (cond
    [(Circle? shape)
     (* pi (expt (Circle-radius shape) 2))]
    [else (expt (Square-side-length shape) 2)]))
```

Note: doesn't actually create a new type, just makes a new type name

Compare: `define-struct` and `U` actually make new types

Type Definitions

```
(define-type Bagel-Flavor String)
(define-type Name String)
(:  favorite-flavor :  Name -> Bagel-Flavor)

(define-type Phonebook (Name -> String))
(define-type Email-Address-Book (Name -> String))
(define-type Address-Book (Name -> String))

(define-type Directory
  (U Phonebook Email-Address-Book Address-Book))

(:  get-contact-info :  Directory Name -> String)
```

Functions on Types

`Listof` and `U` are functions on types: take in types, and output new types

```
(Listof String)
```

`->` is also a function on types. These are equivalent:

```
(: my-func : Integer -> Boolean)
```

```
(: my-func : (-> Integer Boolean))
```

`define-struct` defines a new type AND gives it a name

Type Definitions

Some style guidelines:

- Variable/function names are lowercase
- Type names are uppercase
- Use hyphens when names are more than word
- For functions, use ? when the output is Boolean
- For a function converting between types, use an ->

```
(:  foo :  (Listof String))
(define foo (list "cheez" "its"))
(:  my-really-long-function :  Real -> Real)
(define (my-really-long-function x)
  (+ x 1))

(define-type Bar (String -> Real))
(define-type My-Really-Long-Type (Listof String))
```

Union Types of Symbols

Symbols are special: they are both types and values

When you only have a few cases for a `Symbol`, make a union type for it:

```
(:  define-type Color (U 'red 'blue 'green))
```

```
(:  define-type Dir (U 'north 'east 'south 'west))
```

```
(:  define-type Status (U 'awake 'asleep 'unsure))
```

Putting it together

All data has types:

- `Point, Vector, Square, Circle, Shape, ...`
- `Student, Teacher, Class, Gradebook, ...`
- `User, Item, Cart, Warehouse, Delivery-Truck, Sale, Schedule, Profit-Model, ...`

Typically, the first step in a coding project is to define your types and figure out what functions you want

These are called the *data definitions* or *interface*

Playing Cards

```
(define-type Suit (U 'diamond 'heart 'club 'spade))

(define-struct Card
  ([suit :  Suit]
   [rank :  Integer]))

(define-type Deck (Listof Card))
```

Playing Cards

If we make new types, we should also make new functions:

```
(: red? : Card -> Boolean)
(: black? : Card -> Boolean)
(: face? : Card -> Boolean)
(: card=? : Card Card -> Boolean)
(: card->string : Card -> String)

(: complete-deck : Deck)
(: select : Deck Integer -> Card)
(: complete? : Deck -> Boolean)
(: shuffle : Deck -> Deck)
(: deck=? : Deck Deck -> Boolean)
(: deck->string : Deck -> String)
```

What to know

- Point and define-struct
- Union types
- define-type
- How to implement an interface