CS4450/7450
Chapter 4: Syntax in Functions
Principles of Programming Languages

Dr. William Harrison

University of Missouri

September 11, 2016

What is a Pattern?


```
data I = A | B | C
foo :: I -> String
foo A = "One"
foo B = "Two"
foo C = "Three"
```

A *pattern* is anything in the argument position of a function definition.

What is a Pattern?


```
data I = A | B | C
foo :: I -> String
foo A = "One"
foo B = "Two"
foo C = "Three"
```

A *pattern* is anything in the argument position of a function definition. There are:

- variable patterns, wildcard patterns, constructor patterns, as-patterns

...and bigger patterns are composed of smaller patterns.

Wildcard Patterns

The underscore "_" is a wildcard pattern. They match anything.

```
first :: (a, b, c) -> a
first (x, _, _) = x

second :: (a, b, c) -> b
second (_, y, _) = y

third :: (a, b, c) -> c
third (_, _, z) = z
```

Wildcards are good to use to indicate that you don't care about the value it matches.

Variable Patterns

Variable patterns match anything:

```
addVectors :: (Num a) => (a, a) -> (a, a) -> (a, a)
addVectors a b = (fst a + fst b, snd a + snd b)
```

Variable Patterns

Variable patterns match anything:

```
addVectors :: (Num a) => (a, a) -> (a, a) -> (a, a)
addVectors a b = (fst a + fst b, snd a + snd b)
```

In the following application, `a` and `b` are bound to `(5,6)` and `(7,8)`, respectively.

```
addVectors (5,6) (7,8)
```

Variable Patterns

Variable patterns match anything:

```
addVectors :: (Num a) => (a, a) -> (a, a) -> (a, a)
addVectors a b = (fst a + fst b, snd a + snd b)
```

In the following application, `a` and `b` are bound to `(5,6)` and `(7,8)`, respectively.

```
addVectors (5,6) (7,8)
```

Can also express structure of the input directly using patterns:

```
addVectors :: (Num a) => (a, a) -> (a, a) -> (a, a)
addVectors (x1, y1) (x2, y2) = (x1 + x2, y1 + y2)
```

Constructor Patterns

Recall that lists have two constructors:

```
data [a] = [] | (a : [a])
```

Constructor Patterns

Recall that lists have two constructors:

```
data [a] = [] | (a : [a])
```

Constructors, when appearing in argument position, are patterns:

```
length :: (Num b) => [a] -> b
length []     = 0
length (_:xs) = 1 + length xs
```

Composite Patterns

Patterns can be composed to make bigger patterns, thereby giving you more expressiveness in matching values:

```
tell :: (Show a) => [a] -> String
tell []       = "The list is empty"
tell (x:[])   = "The list has one element: " ++ show
    x
tell (x:y:[]) = "The list has two elements: " ++ show
     x ++ " and " ++ show y
tell (x:y:_)  = "This list is long. The first two
    elements are: " ++ show x ++ " and " ++ show y
```