An Introduction to Functional Programming

Glenn R. Fisher

October 22, 2015

Outline

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

What is Functional Programming?

What is Functional Programming?

Functional programming is a different way to think about writing programs.

What is Functional Programming?

Functional programming is a different way to think about writing programs.

- **First-Class Functions**: A function can return another function or accept functions as parameters.

What is Functional Programming?

Functional programming is a different way to think about writing programs.

- **First-Class Functions**: A function can return another function or accept functions as parameters.
- **Lack of State**: There are no assignment statements. Everything is immutable.

What is Functional Programming?

Functional programming is a different way to think about writing programs.

- **First-Class Functions**: A function can return another function or accept functions as parameters.
- **Lack of State**: There are no assignment statements. Everything is immutable.
- **Expressions (Not Instructions)**: Functions compute results instead of performing actions.

What is Functional Programming?

Functional programming is a different way to think about writing programs.

- **First-Class Functions**: A function can return another function or accept functions as parameters.
- **Lack of State**: There are no assignment statements. Everything is immutable.
- **Expressions (Not Instructions)**: Functions compute results instead of performing actions.
- **Comprehensive Type System**: Create types and catch errors at compile time.

IBM Mobile Innovation Lab

What is Functional Programming?

Functional programming is a different way to think about writing programs.

- **First-Class Functions**: A function can return another function or accept functions as parameters.
- **Lack of State**: There are no assignment statements. Everything is immutable.
- **Expressions (Not Instructions)**: Functions compute results instead of performing actions.
- **Comprehensive Type System**: Create types and catch errors at compile time.

During this talk, we'll try to develop **intuition** behind these ideas.

IBM Mobile Innovation Lab

Why Learn Functional Programming?

Why Learn Functional Programming?

What are the benefits of thinking and programming functionally?

Why Learn Functional Programming?

What are the benefits of thinking and programming functionally?

- Programs are easier to **understand**.

Why Learn Functional Programming?

What are the benefits of thinking and programming functionally?

- Programs are easier to **understand**.
- Shorter, cleaner, and more **maintainable** code.

Why Learn Functional Programming?

What are the benefits of thinking and programming functionally?

- Programs are easier to **understand**.
- Shorter, cleaner, and more **maintainable** code.
- Fewer errors. Higher **reliability**.

IBM Mobile Innovation Lab

Why Learn Functional Programming?

What are the benefits of thinking and programming functionally?

- Programs are easier to **understand**.
- Shorter, cleaner, and more **maintainable** code.
- Fewer errors. Higher **reliability**.
- Excellent **performance** with easy parallelism and concurrency.

IBM Mobile Innovation Lab

Where is Functional Programming Used?

IBM Mobile Innovation Lab

Where is Functional Programming Used?

Companies Using Functional Languages:

- Twitter
- Jane Street Capital
- Airbnb
- Intel
- LinkedIn
- Foursquare
- AT&T
- Bank of America
- NVIDIA
- And Many More...

IBM Mobile Innovation Lab

Where is Functional Programming Used?

Companies Using Functional Languages:

- Twitter
- Jane Street Capital
- Airbnb
- Intel
- LinkedIn
- Foursquare
- AT&T
- Bank of America
- NVIDIA
- And Many More...

You don't need to use a functional programming language to think and program functionally.

IBM Mobile Innovation Lab

Where is Functional Programming Used?

Companies Using Functional Languages:

- Twitter
- Jane Street Capital
- Airbnb
- Intel
- LinkedIn
- Foursquare
- AT&T
- Bank of America
- NVIDIA
- And Many More...

You don't need to use a functional programming language to think and program functionally.

Functional Programming != Functional Programming Languages

IBM Mobile Innovation Lab

Where is Functional Programming Used?

Companies Using Functional Languages:

- Twitter
- Jane Street Capital
- Airbnb
- Intel
- LinkedIn
- Foursquare
- AT&T
- Bank of America
- NVIDIA
- And Many More...

You don't need to use a functional programming language to think and program functionally.

> Functional Programming != Functional Programming Languages
> (Although functional programming languages force you to think functionally.)

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Variables

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Variables

```haskell
x :: Integer
x = 3
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Variables

```haskell
x :: Integer
x = 3

pi :: Double
pi = 3.1415926
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Variables

```
x :: Integer
x = 3

pi :: Double
pi = 3.1415926

b1, b2 :: Bool
b1 = True
b2 = False
```

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Arithmetic

Rapid Introduction to Haskell: Arithmetic

```
ex01 = 3 + 2
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Arithmetic

```
ex01 = 3 + 2
ex02 = 19 - 27
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Arithmetic

```
ex01 = 3 + 2
ex02 = 19 - 27
ex03 = 2.35 * 8.6
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Arithmetic

```
ex01 = 3 + 2
ex02 = 19 - 27
ex03 = 2.35 * 8.6
ex04 = 8.7 / 3.1
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Arithmetic

```
ex01 = 3 + 2
ex02 = 19 - 27
ex03 = 2.35 * 8.6
ex04 = 8.7 / 3.1
ex05 = div 12 5
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Arithmetic

```
ex01 = 3 + 2
ex02 = 19 - 27
ex03 = 2.35 * 8.6
ex04 = 8.7 / 3.1
ex05 = div 12 5
ex06 = 7 ^ 222
```

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: A Note on Parentheses

Rapid Introduction to Haskell: A Note on Parentheses

Many languages use parentheses to call functions.

Rapid Introduction to Haskell: A Note on Parentheses

Many languages use parentheses to call functions.

```
sum(3, 4, 5);
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: A Note on Parentheses

Many languages use parentheses to call functions.

```
sum(3, 4, 5);
```

In Haskell, we omit the parentheses and use spaces instead.

Rapid Introduction to Haskell: A Note on Parentheses

Many languages use parentheses to call functions.

```
sum(3, 4, 5);
```

In Haskell, we omit the parentheses and use spaces instead.

```
sum 3 4 5
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: A Note on Parentheses

Many languages use parentheses to call functions.

```
sum(3, 4, 5);
```

In Haskell, we omit the parentheses and use spaces instead.

```
sum 3 4 5
```

However, we may need parentheses to compute an argument before calling the function.

Rapid Introduction to Haskell: A Note on Parentheses

Many languages use parentheses to call functions.

```
sum(3, 4, 5);
```

In Haskell, we omit the parentheses and use spaces instead.

```
sum 3 4 5
```

However, we may need parentheses to compute an argument before calling the function.

```
sum (1 + 2) 4 5
```

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

```
ex07 = True && False
```

Rapid Introduction to Haskell: Boolean Logic

```
ex07 = True && False
ex08 = not (True || False)
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Boolean Logic

```
ex07 = True && False
ex08 = not (True || False)
ex09 = ('a' == 'a')
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Boolean Logic

```
ex07 = True && False
ex08 = not (True || False)
ex09 = ('a' == 'a')
ex10 = (16 /= 3)
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Boolean Logic

```
ex07 = True && False

ex08 = not (True || False)

ex09 = ('a' == 'a')

ex10 = (16 /= 3)

ex11 = "Haskell" > "C++"
```

IBM Mobile Innovation Lab

Outline

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Functions

Rapid Introduction to Haskell: Functions

```haskell
doubleMe :: Integer -> Integer
doubleMe x = x + x
```

Rapid Introduction to Haskell: Functions

```haskell
doubleMe :: Integer -> Integer
doubleMe x = x + x

quadrupleMe :: Integer -> Integer
quadrupleMe x = doubleMe (doubleMe x)
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Functions

We can also write functions with local variables.

Rapid Introduction to Haskell: Functions

We can also write functions with local variables.

```
hypotenuse :: Double -> Double -> Double
hypotenuse length width = sqrt squaredHypotenuse
  where squaredHypotenuse = length^2 + width^2
```

IBM Mobile Innovation Lab

We can also write functions with local variables.

```
hypotenuse :: Double -> Double -> Double
hypotenuse length width = sqrt squaredHypotenuse
  where squaredHypotenuse = length^2 + width^2

hypotenuse2 :: Double -> Double -> Double
hypotenuse2 length width =
  let squaredHypotenuse = length^2 + width^2 in
  sqrt squaredHypotenuse
```

Outline

Rapid Introduction to Haskell: Types

Rapid Introduction to Haskell: Types

```haskell
type Position = (Double, Double)
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Types

```haskell
type Position = (Double, Double)

position :: Position
position = (1.25, 2.75)
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Types

```haskell
type Position = (Double, Double)

position :: Position
position = (1.25, 2.75)

magnitude :: Position -> Double
magnitude (x, y) = sqrt (x * x + y * y)
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Types

```haskell
type Position = (Double, Double)

position :: Position
position = (1.25, 2.75)

magnitude :: Position -> Double
magnitude (x, y) = sqrt (x * x + y * y)
```

(We will return to this example later.)

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Types

A Person has a name, age, and favorite color.

A Person has a name, age, and favorite color.

```
type Name = String
```

Rapid Introduction to Haskell: Types

A Person has a name, age, and favorite color.

```
type Name = String
```

```
type Age = Integer
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Types

A Person has a name, age, and favorite color.

```
type Name = String
type Age = Integer
type Color = String
```

IBM Mobile Innovation Lab

A Person has a name, age, and favorite color.

```
type Name = String

type Age = Integer

type Color = String

data Person = Person Name Age Color
```

Rapid Introduction to Haskell: Types

A Person has a name, age, and favorite color.

```
type Name = String

type Age = Integer

type Color = String

data Person = Person Name Age Color

glenn :: Person
glenn = Person "Glenn R. Fisher" 22 "Green"
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Types

A Person has a name, age, and favorite color.

```
type Name = String

type Age = Integer

type Color = String

data Person = Person Name Age Color

glenn :: Person
glenn = Person "Glenn R. Fisher" 22 "Green"

favoriteColor :: Person -> Color
favoriteColor (Person name age color) = color
```

IBM Mobile Innovation Lab

Rapid Introduction to Haskell: Types

A Person has a name, age, and favorite color.

```
type Name = String

type Age = Integer

type Color = String

data Person = Person Name Age Color

glenn :: Person
glenn = Person "Glenn R. Fisher" 22 "Green"

favoriteColor :: Person -> Color
favoriteColor (Person name age color) = color

favoriteColor glenn == "Green"
```

IBM Mobile Innovation Lab

Position

Let's create a Position type and write associated functions.

Position

Let's create a Position type and write associated functions.

```
{- a Distance is a length in space -}
type Distance = Double
```

Position

Let's create a Position type and write associated functions.

```
{- a Distance is a length in space -}
type Distance = Double

{- a Position is a pair of x and y distances -}
type Position = (Distance, Distance)
```

IBM Mobile Innovation Lab

Position

Let's create a Position type and write associated functions.

```
{- a Distance is a length in space -}
type Distance = Double

{- a Position is a pair of x and y distances -}
type Position = (Distance, Distance)

{- reflect returns a new Position reflected over the origin -}
reflect :: Position -> Position
reflect (x, y) = (-x, -y)
```

IBM Mobile Innovation Lab

Position

Let's create a Position type and write associated functions.

```
{- a Distance is a length in space -}
type Distance = Double

{- a Position is a pair of x and y distances -}
type Position = (Distance, Distance)

{- reflect returns a new Position reflected over the origin -}
reflect :: Position -> Position
reflect (x, y) = (-x, -y)

{- magnitude returns a Position's distance from the origin -}
magnitude :: Position -> Distance
magnitude (x, y) = sqrt (x * x + y * y)
```

IBM Mobile Innovation Lab

Position

Let's create a Position type and write associated functions.

```
{- a Distance is a length in space -}
type Distance = Double

{- a Position is a pair of x and y distances -}
type Position = (Distance, Distance)

{- reflect returns a new Position reflected over the origin -}
reflect :: Position -> Position
reflect (x, y) = (-x, -y)

{- magnitude returns a Position's distance from the origin -}
magnitude :: Position -> Distance
magnitude (x, y) = sqrt (x * x + y * y)

{- translate returns a new Position translated by an offset -}
translate :: Position -> Position -> Position
translate (x, y) (offsetX, offsetY) = (x + offsetX, y + offsetY)
```

IBM Mobile Innovation Lab

Example: Region

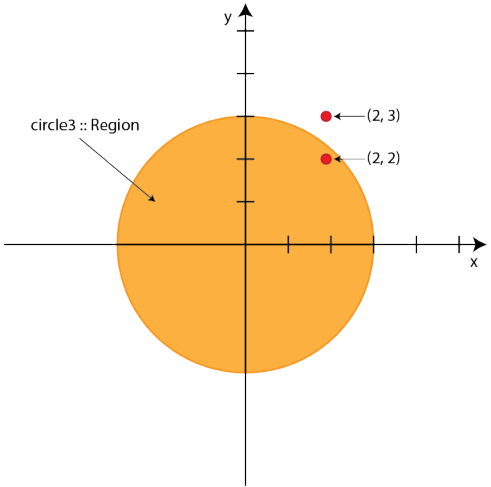
Let's create a Region type and write associated functions.

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?

Example: Region

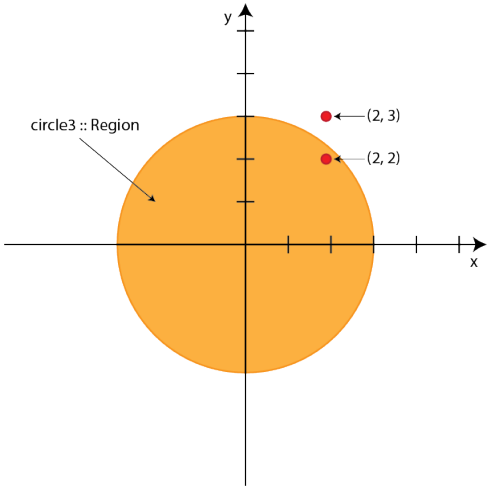
Let's create a Region type and write associated functions.
How should we represent a region in 2D space?



IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?

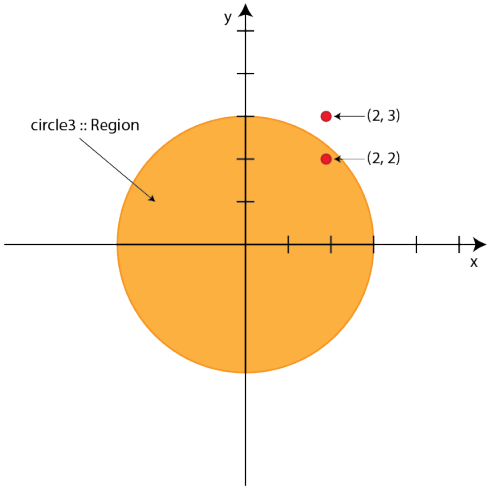



```
type Region = Position -> Bool
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?



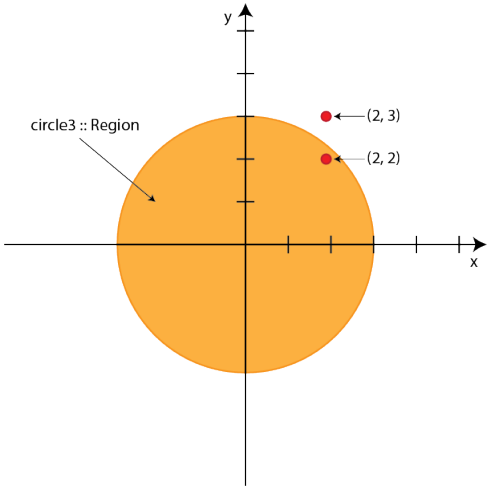

```
type Region = Position -> Bool
```

```
circle3 (2, 2) == True
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?

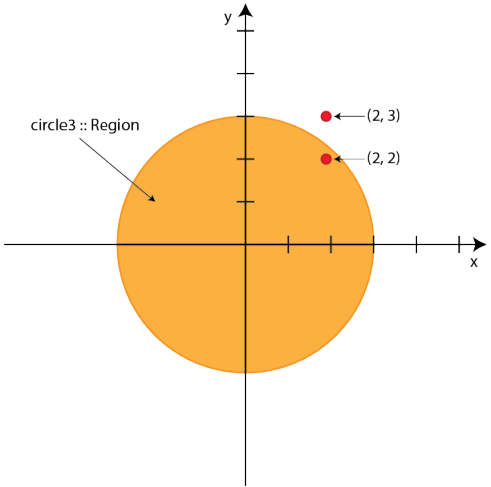



```
type Region = Position -> Bool

circle3 (2, 2) == True

circle3 (2, 3) == False
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?

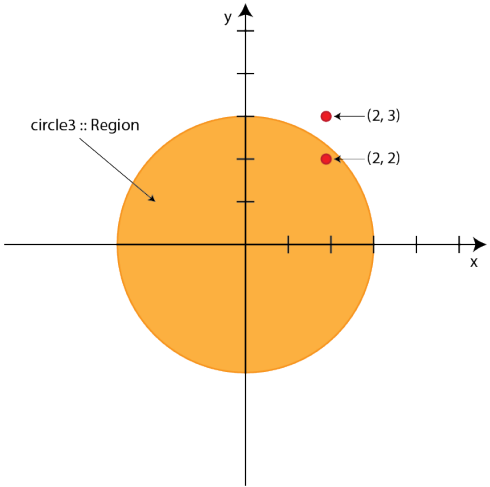



```
type Region = Position -> Bool

circle3 (2, 2) == True

circle3 (2, 3) == False

circle3 :: Region
circle3 position = (magnitude position <= 3.0)
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?

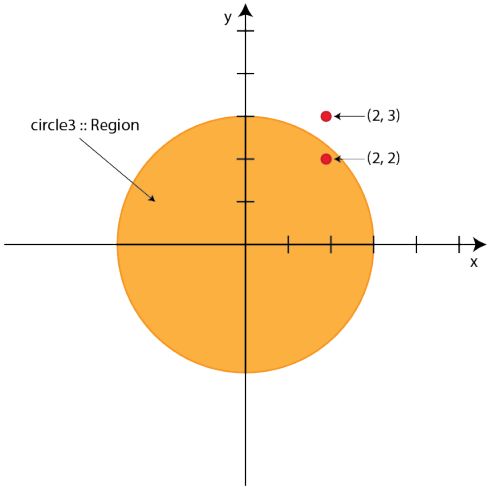



```
type Region = Position -> Bool

circle3 (2, 2) == True

circle3 (2, 3) == False

circle3 :: Region
circle3 position = (magnitude position <= 3.0)

circle4 :: Region
circle4 position = (magnitude position <= 4.0)
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?

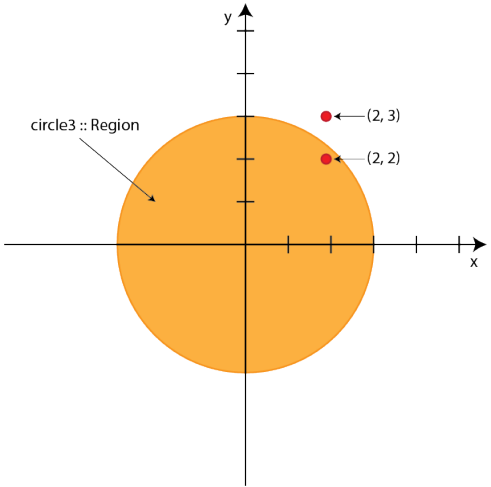



```
type Region = Position -> Bool

circle3 (2, 2) == True

circle3 (2, 3) == False

circle3 :: Region
circle3 position = (magnitude position <= 3.0)

circle4 :: Region
circle4 position = (magnitude position <= 4.0)

circle :: Distance -> Region
circle radius = fun
  where fun position = (magnitude position <= radius)
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.
How should we represent a region in 2D space?




```
type Region = Position -> Bool

circle3 (2, 2) == True

circle3 (2, 3) == False

circle3 :: Region
circle3 position = (magnitude position <= 3.0)

circle4 :: Region
circle4 position = (magnitude position <= 4.0)

circle :: Distance -> Region
circle radius = fun
  where fun position = (magnitude position <= radius)

easyCircle3 = circle 3.0
easyCircle4 = circle 4.0
```

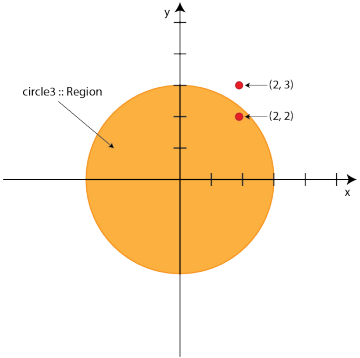
IBM Mobile Innovation Lab

Example: Region

What if we don't want our circle to be centered at the origin?

Example: Region

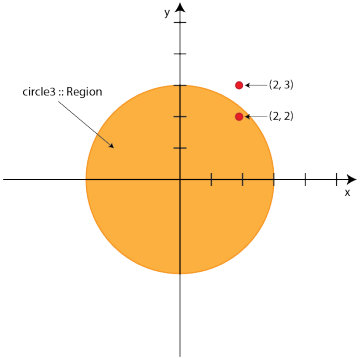
What if we don't want our circle to be centered at the origin?

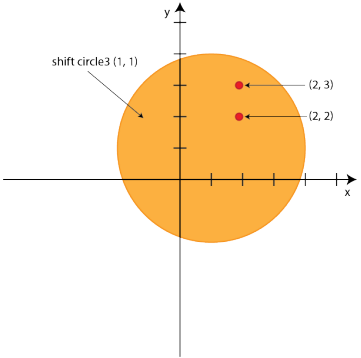


IBM Mobile Innovation Lab

Example: Region

What if we don't want our circle to be centered at the origin?

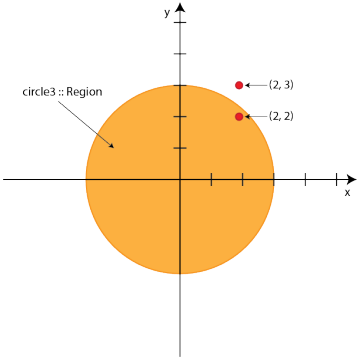


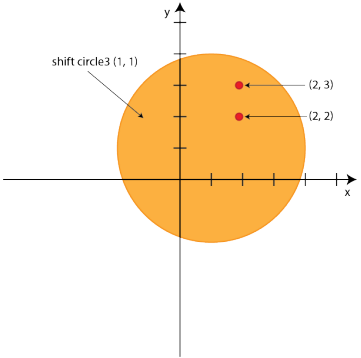


IBM Mobile Innovation Lab

Example: Region

What if we don't want our circle to be centered at the origin?



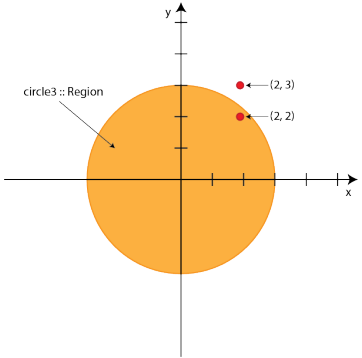


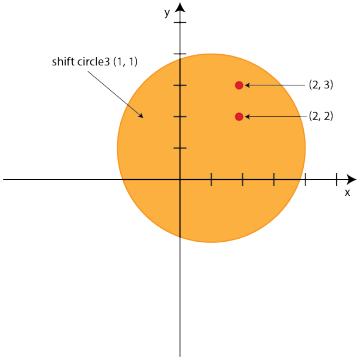

```
{- shift transforms a region by translating it by an offset -}
shift :: Region -> Position -> Region
```

IBM Mobile Innovation Lab

Example: Region

What if we don't want our circle to be centered at the origin?






```haskell
{- shift transforms a region by translating it by an offset -}
shift :: Region -> Position -> Region
shift region offset = fun
  where fun position = region (translate position (reflect offset))
```

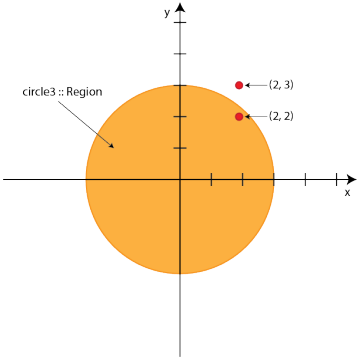
IBM Mobile Innovation Lab

Example: Region

What if we want the region outside of the circle?

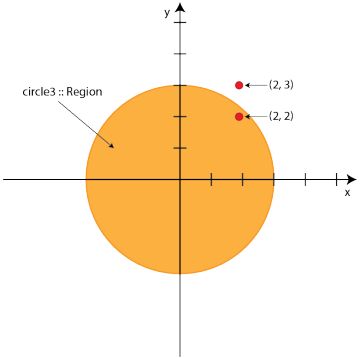
Example: Region

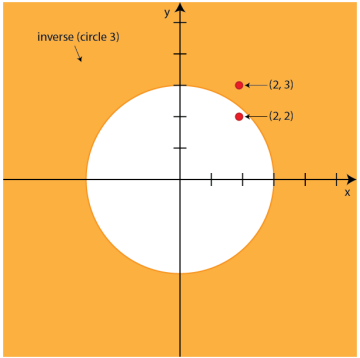
What if we want the region outside of the circle?



Example: Region

What if we want the region outside of the circle?

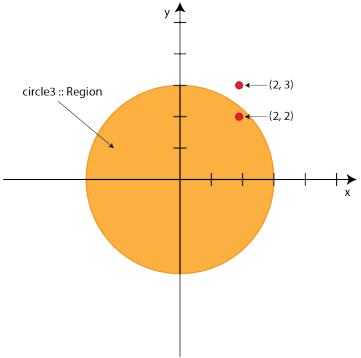


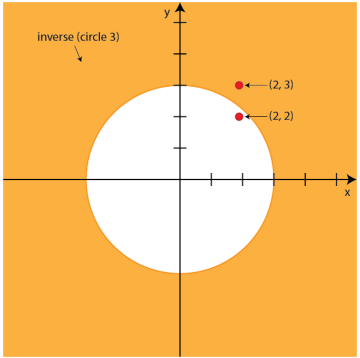


IBM Mobile Innovation Lab

Example: Region

What if we want the region outside of the circle?



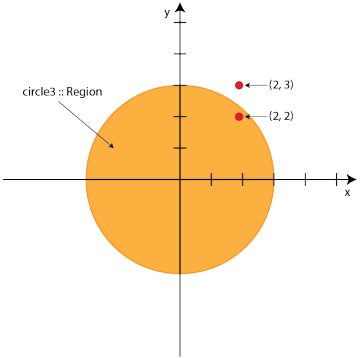


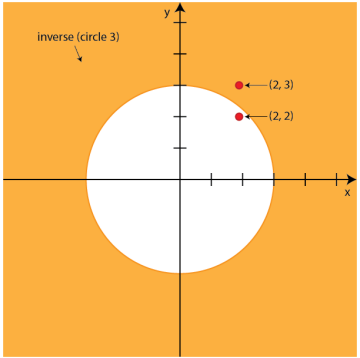

```
{- invert transforms a region by inverting the set of Positions that it contains -}
invert :: Region -> Region
```

IBM Mobile Innovation Lab

Example: Region

What if we want the region outside of the circle?






```
{- invert transforms a region by inverting the set of Positions that it contains -}
invert :: Region -> Region
invert region = fun
    where fun position = not (region position)
```

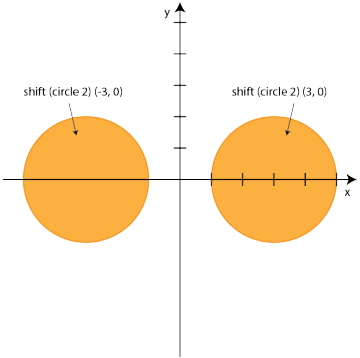
IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?

Example: Region

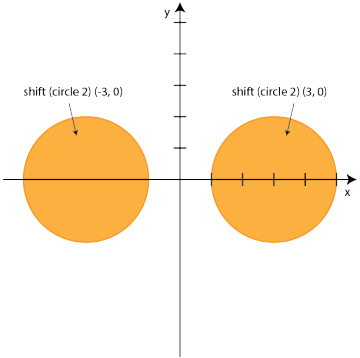
Can we combine regions to create new regions?

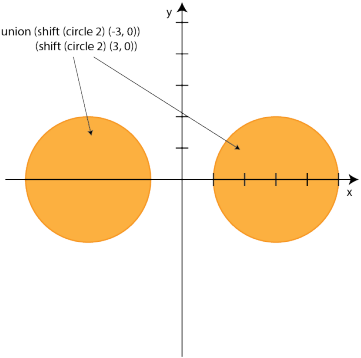


IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?

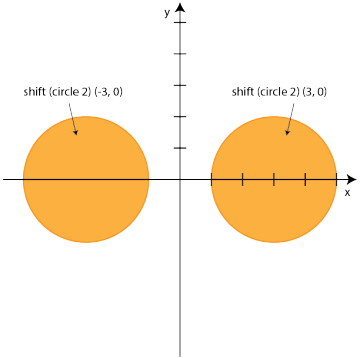


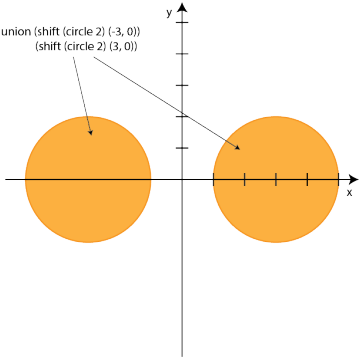


IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?



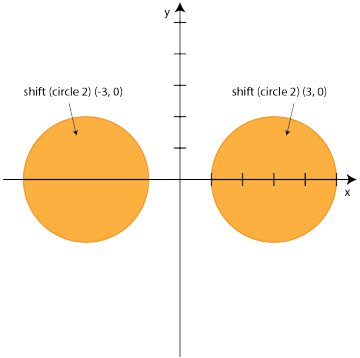


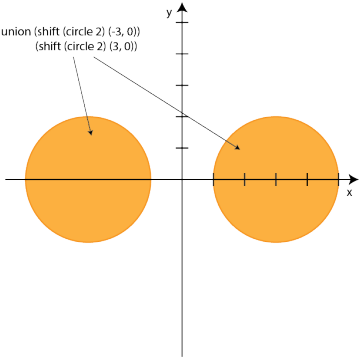

```
{- union constructs a new Region from the union of two Regions -}
union :: Region -> Region -> Region
```

IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?






```
{- union constructs a new Region from the union of two Regions -}
union :: Region -> Region -> Region
union region1 region2 = fun
    where fun position = (region1 position) || (region2 position)
```

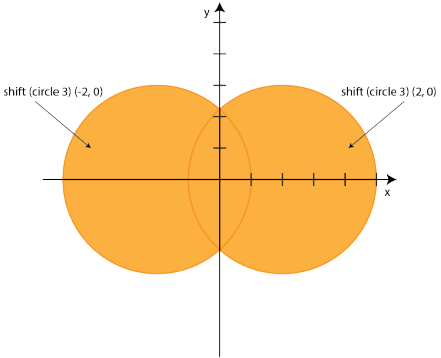
IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?

Example: Region

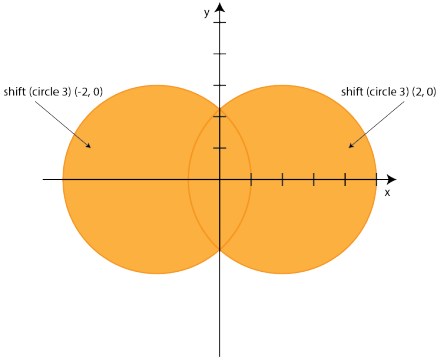
Can we combine regions to create new regions?

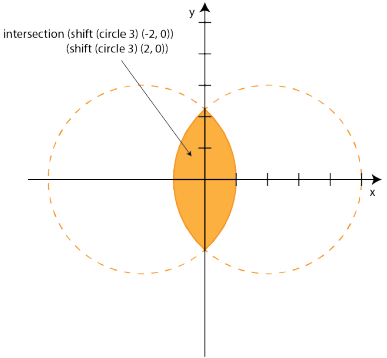


IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?

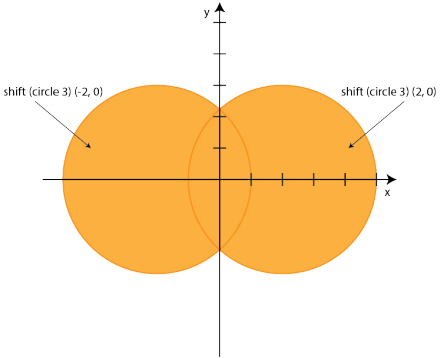


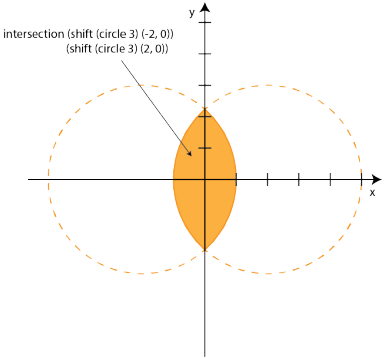


IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?



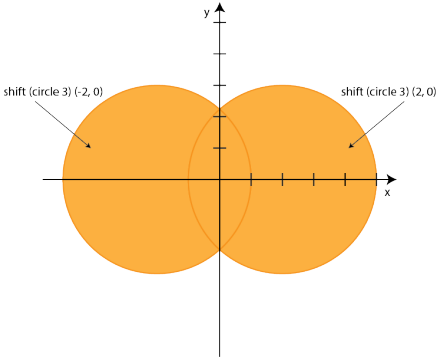


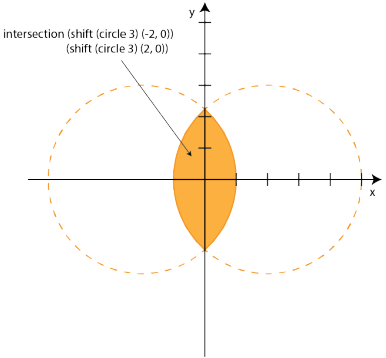

```
{- intersection constructs a new Region from the intersection of two Regions -}
intersection :: Region -> Region -> Region
```

IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?






```
{- intersection constructs a new Region from the intersection of two Regions -}
intersection :: Region -> Region -> Region
intersection region1 region2 = fun
    where fun position = (region1 position) && (region2 position)
```

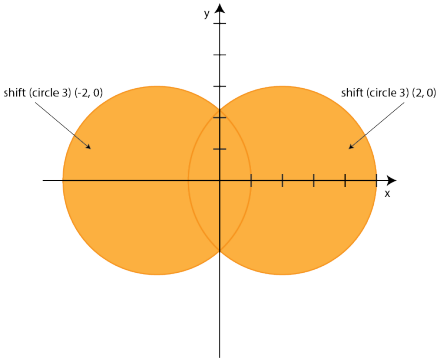
IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?

Example: Region

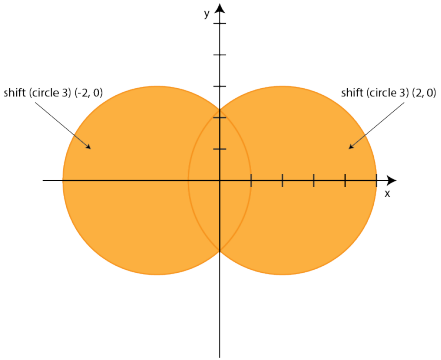
Can we combine regions to create new regions?

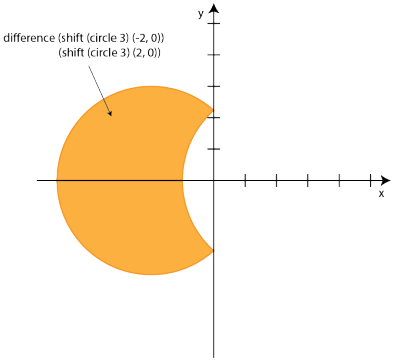


IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?

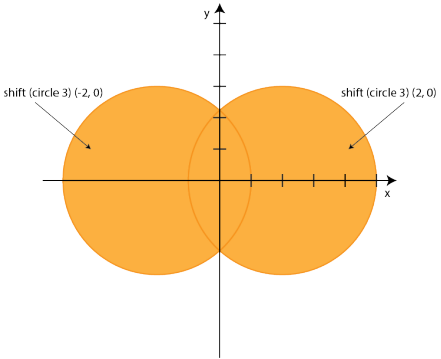


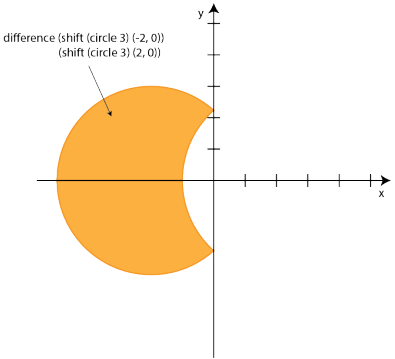


IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?



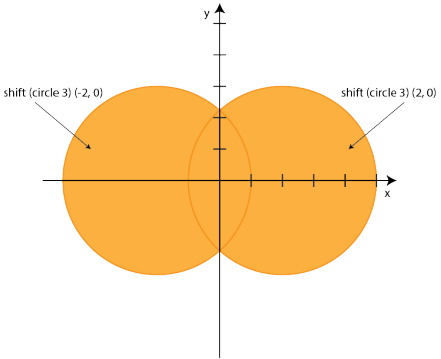


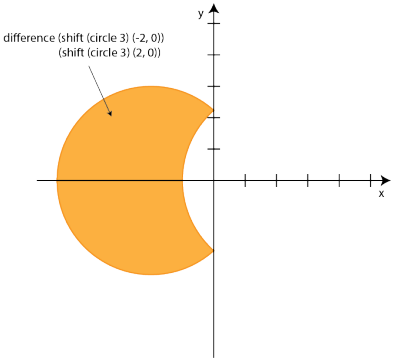

```
{- difference constructs a new Region containing all the Positions
   of the first region that are not members of the second Region -}
difference :: Region -> Region -> Region
```

IBM Mobile Innovation Lab

Example: Region

Can we combine regions to create new regions?






```
{- difference constructs a new Region containing all the Positions
   of the first region that are not members of the second Region -}
difference :: Region -> Region -> Region
difference region minus =
    intersection region (invert minus)
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.

Example: Region

Let's create a Region type and write associated functions.

```
{- A Region is a set of Positions and is defined by a function that
   determines whether a given Position is a member of the set -}
type Region = Position -> Bool
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.

```
{- A Region is a set of Positions and is defined by a function that
   determines whether a given Position is a member of the set -}
type Region = Position -> Bool

{- circle returns a circular Region with the given radius, centered at the origin -}
circle :: Distance -> Region
circle radius = fun
    where fun position = (magnitude position <= radius)
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.

```
{- A Region is a set of Positions and is defined by a function that
   determines whether a given Position is a member of the set -}
type Region = Position -> Bool

{- circle returns a circular Region with the given radius, centered at the origin -}
circle :: Distance -> Region
circle radius = fun
    where fun position = (magnitude position <= radius)

{- shift transforms a region by translating it by an offset -}
shift :: Region -> Position -> Region
shift region offset = fun
    where fun position = region (translate position (reflect offset))
```

IBM Mobile Innovation Lab

Example: Region

Let's create a Region type and write associated functions.

```
{- A Region is a set of Positions and is defined by a function that
   determines whether a given Position is a member of the set -}
type Region = Position -> Bool

{- circle returns a circular Region with the given radius, centered at the origin -}
circle :: Distance -> Region
circle radius = fun
    where fun position = (magnitude position <= radius)

{- shift transforms a region by translating it by an offset -}
shift :: Region -> Position -> Region
shift region offset = fun
    where fun position = region (translate position (reflect offset))

{- invert transforms a region by inverting the set of Positions that it contains -}
invert :: Region -> Region
invert region = fun
    where fun position = not (region position)
```

IBM Mobile Innovation Lab

Example: Region

```haskell
{- intersection constructs a new Region from the intersection of two Regions -}
intersection :: Region -> Region -> Region
intersection region1 region2 = fun
    where fun position = (region1 position) && (region2 position)
```

IBM Mobile Innovation Lab

Example: Region

```haskell
{- intersection constructs a new Region from the intersection of two Regions -}
intersection :: Region -> Region -> Region
intersection region1 region2 = fun
    where fun position = (region1 position) && (region2 position)

{- union constructs a new Region from the union of two Regions -}
union :: Region -> Region -> Region
union region1 region2 = fun
    where fun position = (region1 position) || (region2 position)
```

IBM Mobile Innovation Lab

Example: Region

```
{- intersection constructs a new Region from the intersection of two Regions -}
intersection :: Region -> Region -> Region
intersection region1 region2 = fun
    where fun position = (region1 position) && (region2 position)

{- union constructs a new Region from the union of two Regions -}
union :: Region -> Region -> Region
union region1 region2 = fun
    where fun position = (region1 position) || (region2 position)

{- difference constructs a new Region containing all the Positions
   of the first region that are not members of the second Region -}
difference :: Region -> Region -> Region
difference region minus =
    intersection region (invert minus)
```

IBM Mobile Innovation Lab

Summary

Functional programming is a different way to think about writing programs.

- **First-Class Functions**: A function can return another function or accept functions as parameters.
- **Lack of State**: There are no assignment statements. Everything is immutable.
- **Expressions (Not Instructions)**: Functions compute results instead of performing actions.
- **Comprehensive Type System**: Create types and catch errors at compile time.

Hopefully this talk helped you develop **intuition** behind these ideas.

IBM Mobile Innovation Lab