



Reasoning about Code

Effective Programming in Scala

Let Us Draw a Smiley!



We can break down the problem into simpler parts:

- ▶ draw the eyes;
- ▶ draw the mouth.

Towards Smiling (1)

Here is how we can draw an eye on an HTML Canvas:

```
def drawEye(x: Double, y: Double): Unit =  
  graphics.beginPath()  
  graphics.arc(x, y, 15, 0, math.Pi * 2)  
  graphics.stroke()  
  graphics.fillStyle = "blue"  
  graphics.fill()
```

Towards Smiling (2)

```
def drawMouth(x: Double, y: Double): Unit =  
    graphics.beginPath()  
    graphics.arc(x, y, 200, math.Pi / 4, 3 * math.Pi / 4)  
    graphics.lineCap = "round"  
    graphics.lineWidth = 10  
    graphics.strokeStyle = "red"  
    graphics.stroke()
```

Towards Smiling (3)

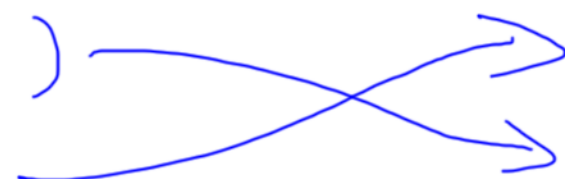
By combining them, we can draw a smiley:

```
drawEye(170, 150)  
drawEye(330, 150)  
drawMouth(250, 200)
```



Puzzle

Are these two variations equivalent? Or do they produce different results?



drawEye(170, 150)
drawEye(330, 150)
drawMouth(250, 200)

drawMouth(250, 200)
drawEye(170, 150)
drawEye(330, 150)

The diagram illustrates a transformation between two code sequences. On the left, the lines are: drawEye(170, 150), drawEye(330, 150), and drawMouth(250, 200). On the right, the lines are: drawMouth(250, 200), drawEye(170, 150), and drawEye(330, 150). Blue arrows show the mapping: the first line maps to the first, the second line maps to the third, and the third line maps to the second, indicating a swap of the last two lines.

Puzzle

Are these two variations equivalent? Or do they produce different results?

```
drawEye(170, 150)  
drawEye(330, 150)  
drawMouth(250, 200)
```



```
drawMouth(250, 200)  
drawEye(170, 150)  
drawEye(330, 150)
```



Interdependencies

```
val graphics = ...
```



```
def drawMouth(x: Double, y: Double): Unit =
```

```
...
```

```
graphics.lineWidth = 10
```

```
graphics.strokeStyle = "red"
```



```
...
```

```
def drawEye(x: Double, y: Double): Unit =
```

```
...
```

```
graphics.stroke()
```

```
...
```



Discussion (1)

The fact that both methods, `drawMouth` and `drawEye`, depend on the same mutable object `graphics` prevents us from reasoning **locally** about these methods.

We can't just expect `drawEye` to do what we read in its implementation, we also have to make sure that `graphics` is in the right state before we call `drawEye`.

Discussion (2)

One strategy to manage complexity is to break down complex programs into smaller programs, and combine them.

This strategy effectively reduces the cognitive load to reason about programs because you only need to reason on a subset of the program.

However, if local reasoning is not possible (ie, if combining programs requires knowledge about their internals), then this strategy breaks down.

Opening the Discussion

How can we prevent this type of error from happening?

