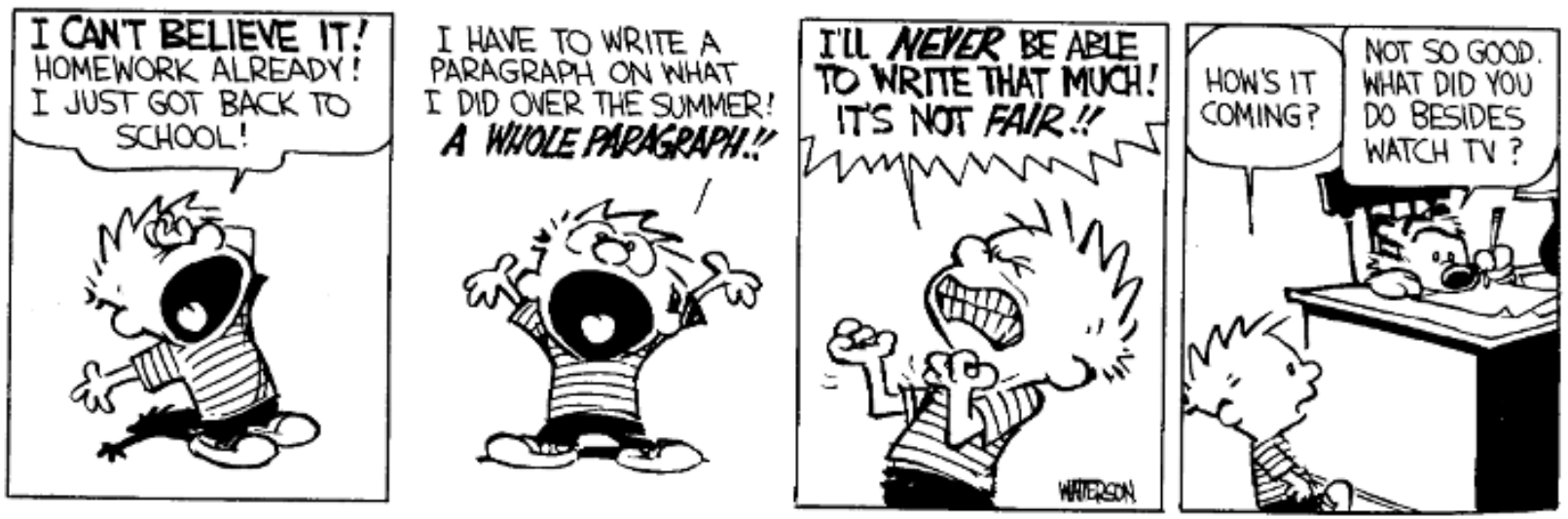
Meeting 03 - Binding and Scope



In-Class Slides

Book Chapter

Reminders

- Laptop-use section: back row
- Use Piazza for all course communication.
  - Energetically engage in discussions with your classmates to help each other on the learning activities — for your Class Participation score.
  - For course administrative things (e.g., grade issues, GitHub access issues), use private messages on Piazza to "Instructors".
- Follow the reading Schedule on the course website.
- Recitation sections are lab sections — bring your laptop!
  - Last Week: Getting your development environment set up!
  - This Week: Finishing HW1

Announcements

- HW1 due Friday 6pm (with the 24-hour grace period for when “stuff happens”)

Monday

Today

- Triage Your Questions
- Preview HW1
- Finish 3.2 Basic Values, Types, and Expressions: A Scala crash course.
- Chapter 4 Binding and Scope: A Scala crash course.

Your Questions?

- Review:
  - What is the *functional* computational model? It was the computational model you learned first, right?
  - What is *referential transparency*?

Your Questions?

Value Bindings

```
1  (2 + 2) + 3
2  (2 + 2) + 6
```

```
res0_0: Int = 7
res0_1: Int = 10
```

Value Environments

How do we evaluate an expression with variable uses?

Value Environments

We need a way of capturing what the variable names in *scope* are bound to — a *value environment*.

Value Environments

Substitution

How do we evaluate an expression with variable uses?

Value Environments: Take-Home Points

- We know how to introduce bindings from variable names to values in Scala (i.e., **`val`**)
- A *value environment* is a map from variable names to values that stores the bindings.
- In order to evaluate an expression containing variable uses, we "apply" a value environment using substitution.
- Conceptually, evaluating a sequence of **`val`** declarations yields a value environment.

Scoping

Shadowing

How do we read this?

```
1  val a = 1
2  val b = 2
3  val c = {
4    val a = 3
5    a + b
6  } + a
```

```
a: Int = 1
b: Int = 2
c: Int = 6
```

Shadowing

Let's pair up and find the binding positions for every variable use in the program below. What is the final environment? Can you *rename* variable bindings and uses consistently to eliminate the shadowing?

```
1  val a = 2
2  val c = {
3    val a = 3
4    val b = a * a
5    a * b
6  } * a
7  val d = {
8    val b = 3
9    a * b
10 } * c
```

```
a: Int = 2
c: Int = 54
d: Int = 324
```

Shadow-Respecting Substitution

What if we “naively” applied substitution of env1: [a ↦ 2] to the rest of the expression?

```
1  val a = 2
2  // env1: [a |-> 2]
3  val c = {
4    val a = 3
5    val b = a * a
6    a * b
7  } * a
8  val d = {
9    val b = 3
10   a * b
11 } * c
```

```
a: Int = 2
c: Int = 54
d: Int = 324
```

Free versus Bound Variables

```
1 {
2   { val x = 3; x + y }
3 }
```

```
cmd10.sc:1: not found: value y
val res10 = { val x = 3; x + y }
                            ^Compilation Failed

:
Compilation Failed
```

Free versus Bound Variables

A *closed* expression is one that has no free variables:

```
1 {
2   { val y = 4; { val x = 3; x + y } }
3 }
```

```
res10: Int = 7
```

Functions

Closures

An expression defining a function can refer to variables bound in an outer scope:

Tuples

Using Tuples