Tips for Teaching Scala

@noelwelsh

noelwelsh.com

underscore

Inner Product

Outline

Teaching Scala
for **~8 years**

Teaching is a
distinct skill

Teaching is an **undervalued skill**

Seniors **teach**
juniors

ScalaBridge **teaches** to increase diversity

We **teach**
ourselves

5 Tips for teaching

5 Further resources

1 Language choice

1 Language choice

2 Curriculum

1 Language choice

2 Curriculum

3 Pedagogy

5 Tips for Teaching

1 **Notional** Machines

Cargo-culting

vs

Understanding

Understanding requires a **simplified** machine model

Substitution

```
val a = 1
val b = a + a
b * b
```

```
val a = 1
val b = a + a
b * b
```

```
val a = 1
val b = 1 + 1
b * b
```

```
val a = 1
val b = 2
b * b
```

```
val a = 1
val b = 2
b * b
```

```
val a = 1
val b = 2
2 * 2
```

```
val a = 1
val b = 2
```

4

For **methods**

For **types**

For **pure code**

Easy!

Easy!

This is the **point**

This is the **point**

# of immutability

This is the **point**

# of monads

This is the **point**

# of FP

2 Programming **Strategies**

Systematic and
repeatable
programming

Similar to **design patterns** but **wider in scope**

10 strategies in total

Illustrate 3

Algebraic data types
Structural recursion
Following the types

Algebraic data types

Structural recursion

Following the types

The data is described using **and** & **or**

If **A** has a **B and C**

final case class **A**(b: **B**, c: **C**)

If **A** is a **B** **or** **C**

sealed trait **A**
final case class **B**() extends **A**
final case class **C**() extends **A**

List is a **Pair** **or** **Empty**

sealed trait **List**
final case class **Pair**() extends **List**
final case class **Empty**() extends **List**

Pair has a **head and** a **tail**

```
sealed trait List
final case class Pair(head, tail)
  extends List
final case class Empty() extends List
```

head has type **A and tail** has type **List[A]**

```
sealed trait List[A]
final case class Pair[A](
  head: A, tail: List[A]) extends
  List[A]
final case class Empty[A]() extends
  List[A]
```

```
sealed trait List[A]
final case class Pair[A](
  head: A, tail: List[A]) extends
  List[A]
final case class Empty[A]() extends
  List[A]
```

(Invariant for simplicity)

Algebraic data types

Structural recursion

Following the types

We are doing **any transformation** on an **algebraic data type**

Pattern matching or **Polymorphism**

Pattern matching or Polymorphism

If **A** is a **B** **or** **C**

```
anA match {
  case B() ⇒ ???

  case C() ⇒ ???

}
```

If **A** has a **B and C**

case **A**(b, c) ⇒ ???

Additionally: where the **data is recursive** the **method is recursive**

```
sealed trait List[A]
final case class Pair[A](
  head: A, tail: List[A]) extends
  List[A]
final case class Empty[A]() extends
  List[A]
```

```
def doSomething = {
  aList match {
    case Empty() ⇒ ???

    case Pair(h, t) ⇒ ??? t.doSomething()

  }
}
```

How do we complete
the right-hand side???

Algebraic data types
Structural recursion
Following the types

Let the **types guide you**
to a solution

What is the goal?
What is available?
Assemble the jigsaw

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    ???
}
```

We're transforming an algebraic data type

Therefore use **structural recursion**

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =
    this match {
      case Empty() ⇒ ???

      case Pair(h, t) ⇒ ???

    }
}
```

Remember the rule for

recursion

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    this match {
      case Empty() ⇒ ???

      case Pair(h, t) ⇒

        ??? t.map(???)
    }
}
```

Now **follow the types**

First consider the **Empty** case

Find the **goal**

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    this match {
      case Empty() ⇒ ???

      case Pair(h, t) ⇒

        ??? t.map(???)
    }
}
```

Goal is **List[B]**

Goal is **List[B]**

What is available?

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    this match {
      case Empty() ⇒ ???

      case Pair(h, t) ⇒

        ??? t.map(???)
    }
}
```

Goal is **List[B]**

Available is **f: A ⇒ B**

Goal is **List[B]**
Available is **f: A ⇒ B**

And the **constructors**

Goal is **List[B]**

Available is **f: A ⇒ B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

Goal is **List[B]**

Available is **f: A ⇒ B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

The only thing we can use is **Empty**

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    this match {
      case Empty() ⇒ Empty()

      case Pair(h, t) ⇒

        ??? t.map(???)
    }
}
```

Now consider the **Pair** case

Goal is **List[B]**

Goal is **List[B]**

What is available?

Goal is **List[B]**

Available is **f: A ⇒ B**

Goal is **List[B]**
Available is **f: A ⇒ B**

And the **constructors**

Goal is **List[B]**

Available is **f: A ⇒ B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

Goal is **List[B]**

Available is **f: A ⇒ B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

And **h**, **t**, and **t.map(???)**

Goal is **List[B]**

Available is **f: A ⇒ B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

and **h: A**, **t: List[A]**,

t.map(???): List[B]

Goal is **List[B]**

Available is **f: A ⇒ B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

and **h: A**, **t: List[A]**,

t.map(???): List[B]

Goal is **List[B]**

Available is **f(h): B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

and **t: List[A]**,

t.map(???): List[B]

Goal is **List[B]**

Available is **f(h): B**

and **Empty: () ⇒ List[A]**

and **Pair: (A, List[A]) ⇒ List[A]**

and **t: List[A]**,

t.map(???): List[B]

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    this match {
      case Empty() ⇒ Empty()

      case Pair(h, t) ⇒

        Pair(f(h), t.map(???))
    }
}
```

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    this match {
      case Empty() ⇒ Empty()

      case Pair(h, t) ⇒

        Pair(f(h), t.map(???))
    }
}
```

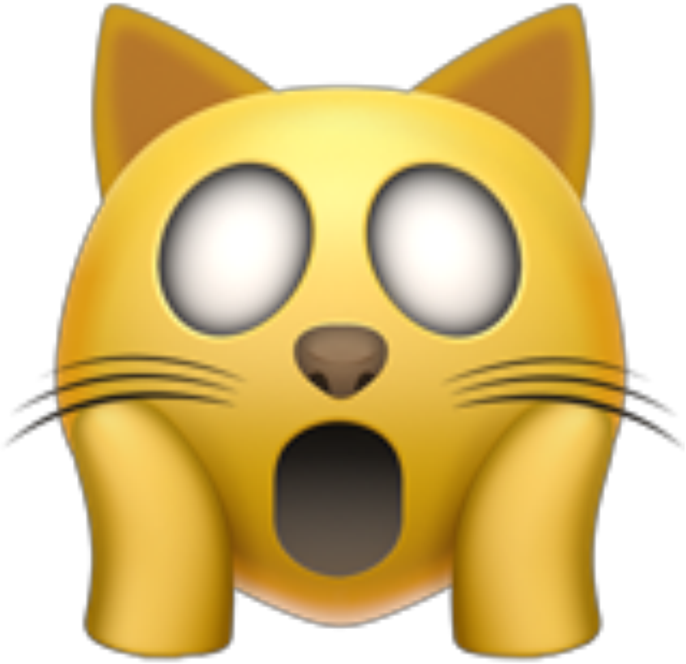
Goal is **A** ⇒ **B**

```
sealed trait List[A] {
  def map[B](f: A ⇒ B): List[B] =

    this match {
      case Empty() ⇒ Empty()

      case Pair(h, t) ⇒

        Pair(f(h), t.map(f))
    }
}
```

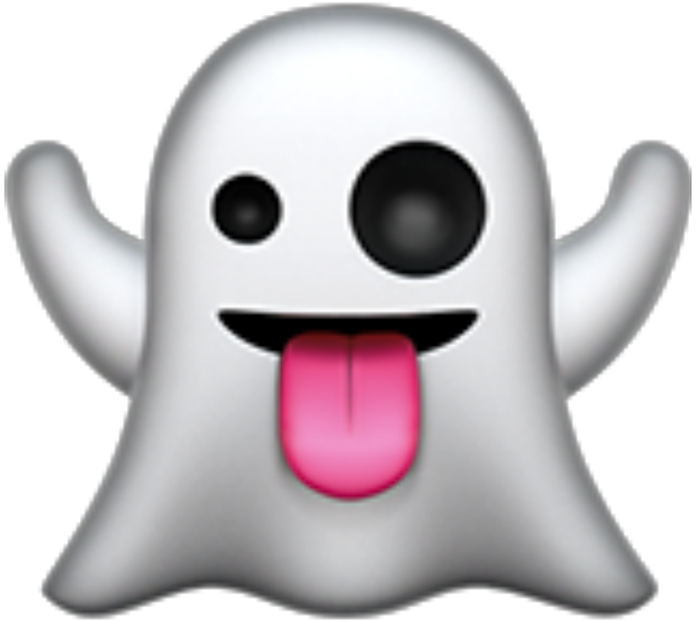
3 It is **more** **than** coding

Debugging and **tool use**
are essential to
programming

scalac says **“found type A but expected type A”**



git says **“you are in a detached head state”**



A **lot** of implicit knowledge

Live coding!

“But I **can’t** think and type!”

That’s the point

Demonstrate **error recovery**

Demonstrate **tool use**

Get **students** to **correct mistakes**!

4 Shut up!

Thinking while listening
is **hard**

But you want to **help**

Give **prompts** for **thinking**

“What **strategy** are **you using**?”

Give **feedback**

“I think this part is wrong.
Can **you** see why?”

Get students to **voice**
their **mental models**

“**Explain** to me what you’re doing here.”

The teachers job is to

present material

The teachers job is to **uncover** and **correct flaws** in their **mental models**

Not to be an **oracle**

5 **Peer** learning

Students can **learn** by
teaching other students

The **best** way to **learn** is to **teach**

Teaching forces **creation** of a **coherent mental model**

Three ways students can teach each other

1 The **Hypothesis Game**

Teacher asks a **question**

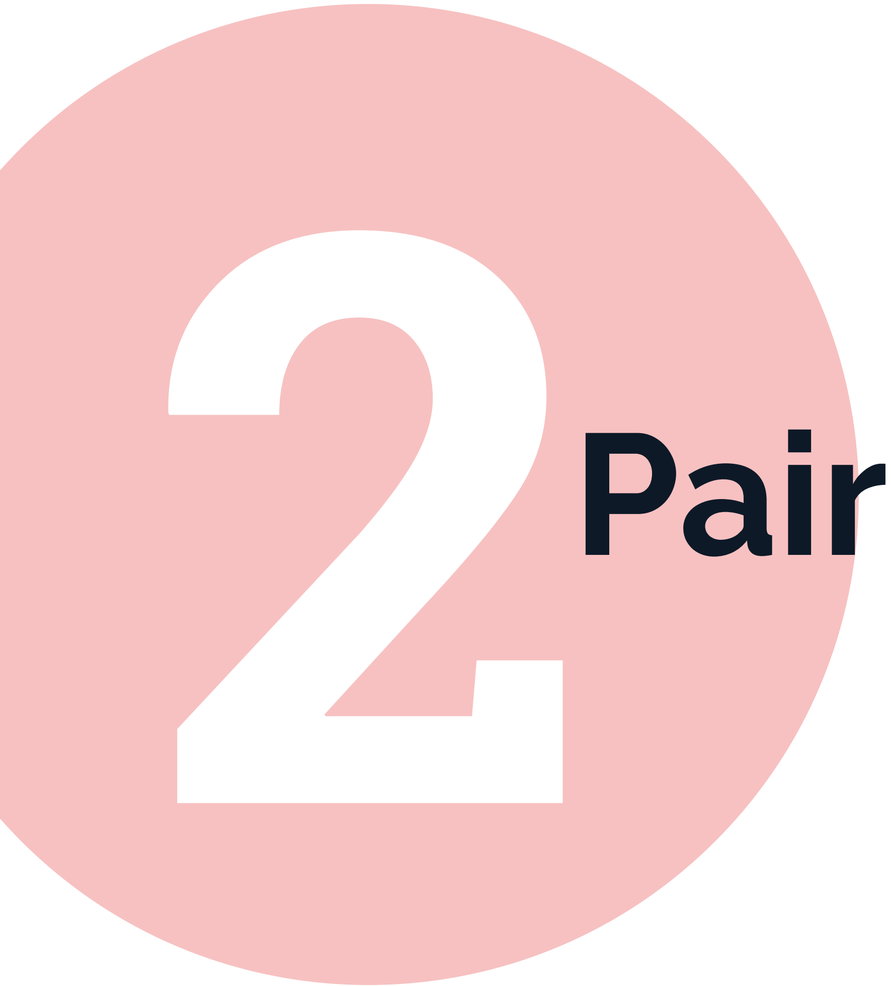
Teacher asks a **question**

Students **answer individually**

Teacher asks a **question**

Students **answer individually**

Students **justify answer** to another student



Pair Programming

Two students **one** keyboard

Many students **one** keyboard

Don’t let any 1 student
bogart the keyboard

Non-typists **must be active**

tuple.app/pair-programming-guide

3 The **Rubber Duck**

Explain to an **inanimate object**

Pair-programming **alone**

5 Further resources

1

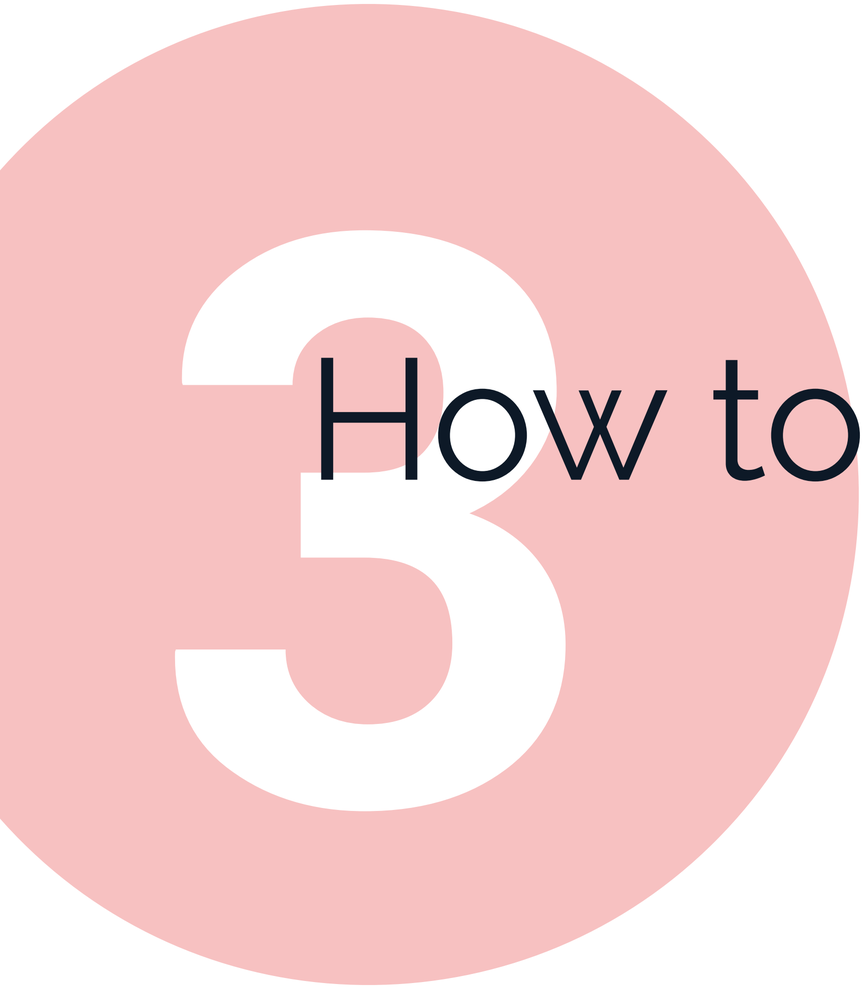
Ten quick tips for teaching programming

https://doi.org/10.1371/journal.pcbi.1006023

2

Teach Together

teachtogether.tech



How to Design Programs

htdp.org

4

Visible Learning

visible-learning.org

5 Your own **practice**

5

Your own **deliberate practice** and **reflection**

Conclusions

Teaching is a **skill**

Teaching is a skill **you can learn**

Teaching is not very different from **learning**

My 5 Tips

1 Notional machine

2 Strategies

3 More than coding

4 Shut up

5 Peer learning

Want to teach?

ScalaBridge needs mentors

scalabridgelondon.org

@noelwelsh

noelwelsh.com

underscore

Inner Product