####– ##### ########### ####

######

######### © 2008 ############ ###. ###. ### ###### ########

########

60 # 75 ###### ############ ## ########### ######## ###–######## ######## ## ######## ########### ##### ### ##### ########### ######## [####://#####–####.###/]. ######## #### ## ##### ######## ### ####### ###### ## ##### ## ### ##### ## ######## ###########, #### ######## #### ########## #### ##### ##### ###### ### ############ ########### ######.

########.

########

<##:#######></##:#######>

Formalities

Questions

- #########. ###### #### ## ### ### ####.
- #######, #### ######### #### #### ## #### #########! ##### #### ## ######## ### ## #### ###########.
- ##### ## ### ##### ##### ### ##### ### ## ######### ######?[1]
- #######, *please see me after this session, or email me with your question.*

Objectives

- #### #### ### ##### #### ####?
- ##### ###### ##########. ### #### #### ######## ##### ##### ### ## #### ##### ## #### # ##### ####–#####.

[1] ### # #### ## ####### #### ####### ### ##### ### #### ##### ## # ##### ###### ######### ##### ### ### ## ## ########## #####, ###### ##### ## #### ## ######## ## ######### ######## # #### ####### ## ### #### ### ###### ## ### ##### ## ## ######## #### ## ### #### ######## ##########. ### ## # #### ## ## ## ## ### ####, ##### ## ######### ###### ## #########, ## ## ## ########### ########### ## ### #### ## ##### ###### ## ##### ##, # ###### ####### ## ####### ## ## ####### ########. ##, #######, ### ######### ## #### # ###### #### ######## ## ####### #####, ###### ## ### ###### ##### ##### ######, ### ######### #### ### ##### ## ######## ## ######, ########## ## ####### ## ### ######### ##### ###### # ##### ## ############ ### ####### ### ####### ## ### ########## ## ### ############ ## ## ########### ### ## ## ### ########## ## ## ####### ####.# # ######## ####### (1872 # 1970)

- ### ## ##### ######### ##### #### #####!
- ###### ## #####–######### ######## ############ ### ##### ### ####### ######## ###–####### ######.
- # ## #### ########## #### ########.[2]
- ## ## ### ## ######### ## ######## ### ## ### ########## ## ### ######## ## #### ## ## ####### ########### ### ########## ##### # ###### ### ## #### ###########.
- #### ########. #### ###### #########.

Keep it DRY

Suppose we had...

```
Iterable<String> convertValueOf(Iterable<Integer> x) {
  Collection<String> y = new LinkedList<String>();
  for(Integer i : x)
    y.add(String.valueOf(i));
  return y;
}

...

Iterable<String> convertForDatabase(Iterable<Integer> x) {
  Collection<String> y = new LinkedList<String>();
  for(Integer i : x)
    y.add(new StringBuilder(i + 7).reverse().toString());
  return y;
}
```

- ###########! ##### ########.

So then we had...

```
interface Stringer {
  String stringify(Integer i);
}

Iterable<String> stringifyConvert(Iterable<Integer> x, Stringer s) {
  Collection<String> y = new LinkedList<String>();
  for(Integer i : x)
    y.add(s.stringify(i));
  return y;
}
```

- #### ## ###### ## ### ### ######### ############### ## `Stringer` ### ### ### ### #####.
- ###...

[2] ## ## ########### ########## ## ##### #### ########### ## ######## #### #### ### # ##### ######## ## #####: ## ######### ########### #### ### ######## ######### ###### #### ## ############.

But then we also had...

```
Iterable<Integer> convertParse(Iterable<String> x) {
  Collection<Integer> y = new LinkedList<Integer>();
  for(String s : x)
    y.add(Integer.parseInt(s));
  return y;
}
```

- ## ##, #### ########### # #### ###########!
- ## ##### ############ #### ### ###### #####

Refactoring Wizardry

```
interface Mapper<A, B> {
  B map(A a);
}

<A, B> Iterable<B> mapperConvert(Iterable<A> x, Mapper<A, B> m) {
  Collection<B> y = new LinkedList<B>();
  for(A a : x)
    y.add(m.map(a));
  return y;
}
```

- ####### ########? `LinkedList`?
- #### ## ## #### ## ### ######### #### ## ### #### ### `Iterable`, ### ## ##### ## ##### ### ## ## ## `Iterable`? #.#. ## ######## #### ######### ### ######### ## ## ###### ##.
- ###### ## ####### ### ### ####### ###### # ### ###### #### ######## ### `map` ######? ## ####### ####### ###### ###### #### ####### ### ### ##########?
- ####### ## #### ## ## # ### ####, #### ## ###### ######## ## # ####### ######## ## ## ###?

The DRY Continuum

[redacted]

A Brief Introduction to Functional Programming

Remember this crazy guy?

2008:

Fundamentalism

- *Fundamentalist*[3] ########## ########### ## ####### ## ######### ####–####### ## ####–######.
- # ####–###### ## ### ######## ##### #### ## ######## #### ######### #.#. #########, #####, ############ #/#.
- ######## `java.lang.String`, ##### ## ### ## #########. ######, ## ### ###### ### ## ### ####### ### *referentially transparent*.
- ########### ############ (## ######) ## ### ####–###### ## ####–####### (## ########).

Referential Transparency

- ### ## ## ######## #### *idempotence*.[4].
- ### ###### ####.

[3] #### ######, #### ########, 2008

[4] ### ########### ###### ########### ############ [####://####.#######.###/###########–######–###########–############/]

- ##:

  ```
  g(f(args));
  g(f(args));
  ```

  ## ##########[5] ##:

  ```
  final T t = f(args);
  g(t);
  g(t);
  ```

  ### ### ######## g ### ### ######### ## f (##### ## #### ###### args) #### ## ### #### f ## ############# ###########.

For example

- ### #######

  ```
  g(s.charAt(i));
  g(s.charAt(i));
  ```

  ## ###### ########## (########## ## g) ##:

  ```
  final char c = s.charAt(i);
  g(c);
  g(c);
  ```

  #########, charAt ## ############# ###########.

- ### ####; new ## # ####–######. #### ###### ## #### #### ########### ### #### ######### ####?

- #### ### ####; #### ##### ## ######### ######### ## ## ### ## ########.[6]

To care or not to care

```
String s1 = ...
String s2 = ...
method(s1.charAt(x), s2.charAt(y));
```

1. s1.charAt(x) ######## #####, #### s2.charAt(y).
2. s2.charAt(y) ######## #####, #### s1.charAt(x).
3. ##### ##### ######## ##### # ###### #### ###.
4. # ##### #### ##### ######## ##### ### # ##### ####.

[5] ### ## ################# ####### #######.
[6] ########## #### [####://#############.###/] –*Quod Erat Demonstrandum*

Which First?

```
char charAt2(int index) {
  writeFile(this, index);
  return this.charAt(index);
}

method(s1.charAt2(x), s2.charAt2(y));
```

1. `s1.charAt2(x)` ######## #####, #### `s2.charAt2(y)`.
2. `s2.charAt2(y)` ######## #####, #### `s1.charAt2(x)`.
3. ##### ##### ######## ##### # ###### #### ###.
4. # ##### #### ##### ######## ##### ### # ##### ####.

So what?

- #### #### ######## ########## ##### ## ####### ####### ########## ## #### ###### ###–########,[7] ##### ###### #### ######## ####### ########## ### #### ###########.
- #### #### ####### ###### (#### ######) ###### ## ########### ### ## ### *essence* ## ####–####### ##########. ###### ## ### ### ####.
- #### #### ## ############# ###### ## ###### #####.
- ## #### ##### ####, ###### ############, ###### ########## ## ########### ######, ### ###### # ######## #### #######. #### ## ####### ### ### ## ###### #### ####?

Where Does Scala Fit?

- ###### ## #######, #### #### ##### ########## (#######, ######–2, #, ####, ##).
- #### ## #######, ###### ## ### ###### (#####, ###, ##).
- #### ## ####### (#######, #####).
- ##### ## ######### ## ### ####[8] ## ########### ### ## ## #### ###### #######.
- ##### ####### ## ## ### ########## ###########, ##### #### ######## ## ######## ###### ####. ##### ## ### # ########## ########### ######## ## ### #### #####, ### ## ## ### ####### ##### (#.#. #####–##### #########).
- ##### ######## ## `.class` ##### ### ############# #### ### ####–######## ####### ## ### ####.
- ##### #### ### ##### ## ## ####### ####–####### ## ## #####!

Basic Syntax of Scala

val,var,def,lazy val

- `var` ######## # ####### ####. ######### ## # ###–`final` ## ####. ##### ### ### ## `var` ###### ## ########### #############.

[7] *(paraphrase)* ### ########## ########### ####### [####://###.####.########.##/#####/######/#####.####], #### ######
[8] ##### ############ ######## ## ########

* `val` ######## ## ######### #### #### ## ######### ## ### ########### #####. ######### ## # `final` ## ####.
* `def` ######## ## ######### #### #### ## ######### #### #### ###. ## ######## #### ### ######. ######### ## # ###### ## ####.
* `lazy val` ######## ## ######### #### #### ## ########### ##### ##### ###.
* ####### *(welcome to the Scala interpreter)*

```
scala> val a = 7
a: Int = 7

scala> def b = 8
b: Int

scala> def c(n: Int) = n + 42
c: (Int)Int

scala> var d = 9
d: Int = 9

scala> lazy val e = 10
e: Int = 10
```

if/else

* ####### `if/else` ######## ###### `if/else` ### #### ####### ######## `?:`.
* #######

```
scala> val t = if(true) 7 else 8
t: Int = 7

scala> if(false) println("foo") else println("bar") // side-effect
bar

scala> println(if(true) "foo" else "bar")
foo
```

First-class Functions

* ##### ### *first-class functions*. #### ####### ## ### #### 7 #### [####://#####.####/] ########.
* #### ####### ### ## ####### ## #### ########## # ##### #### ########### ########## #####:

```
interface Function<A, B> {
  B call(A a);
}

interface DatabaseWalloper {
  Walloped wallop(Connection a);
}
```

Function literal

- ## #####, ####### ## ####### # Function<String, Integer>, ## #### String => Integer ### ####### ## function.call(a) ## ##### function(a).
-

```
scala> def reverseParse(s: String, f: String => Int) = f(s.reverse)
reverseParse: (String,(String) => Int)Int

scala> val c = reverseParse("5632", Integer.parseInt(_))
c: Int = 2365
```

- ##### ######### ### #####-##### ## ### ## ##### ###### ###### #### ####, #### ####### ####.

```
scala> def compose[A, B, C](f: B => C, g: A => B): A => C = (a: A) => f(g(a))
compose: [A,B,C]((B) => C,(A) => B)(A) => C
```

Other Keywords

- trait ## #### # #### interface ### ###### *mixin composition*.

```
scala> trait Foo { def bar(n: Int): String }
defined trait Foo

scala> trait Bar
defined trait Bar

scala> trait Baz extends Foo with Bar { val baz: java.util.LinkedList[Int] }
defined trait Baz
```

- class ## ####### ## # #### class, ####### case class ### ## #### ##########.

```
scala> case class Person(name: String, age: Int)
defined class Person

scala> val p = Person("Bob", 42)
p: Person = Person(Bob,42)

scala> val z = p.name
z: String = Bob
```

- # sealed class ## trait ######### ######## ######## ## ### ######### ###### ####. #### ### ## ###### ########## ## ########## ### #### #### #### ###### ####### ### ###-##### # ##### #### ## ############ ###########.
- sealed ##### ### case class ### #### ## ###### *Algebraic Data Types*.

Type Parameter Variance

- ## ####, # List<String> ## ### # List<CharSequence> #### ###### # String ## # CharSequence.

```
void m1(java.util.List<CharSequence> s) {}
void m2(java.util.List<String> s) { m1(s); } // bzzt!
```

- ###### #### ########## ### ###### *invariant*, ### ##### ## #### ## #### #### *covariant*[9].
- ########## ####### ### #### #### ####–#######[10]
- ### ## ###### ####### ## #### #######! ## ### #### ###########, ## # ##### #### # ##### ## ### ##### ###### ####. ## ### + ## ###### ##########[11]

```
scala> trait List[+A] { val head: A; val tail: List[A] }

scala> val x = new List[String] { val head = "abc"; val tail = null }

scala> val y: List[CharSequence] = x
```

Imports

- ### `import` ####### ## #### ### ######### ########.
- ######## ### ############; `foo.bar` ### ###### ## ########## ## `foo`. #### #### ####### ## ###### ## ######## #### ####. #.#. #### ##### ###### #### ## `java.util.concurrent` ### ###### ## #### ########### (`java.util`) ####### #########.
- ####### ### ####### ######## ####### `import java.util.{LinkedList, HashSet}`.
- ####### ### ######## `import java.util._` ## `import java.util.Collectons._`.
- ####### ### ###### ######## ## # ###### #### ### ### ###### ###########.

Basic Features of Scala

Type Inference

- ## ##### #### ## ########## ####### #### ###########.

[9] #### ## ### ######## #### ## #### # #### #### ## ########## ### ## ########### #### `T extends U` ### ######–######## #### `T super U`.
[10] ### #######, ####–######### ## # ######### #### ##### ####### ## ## `ArrayStoreException`.
[11] ### – ## ###### #############; ####### ######.

Static Verification

- ### ## ### ##### #### ###### ############!

I am always right

```
scala> val t = 7
t: Int = 7

scala> t.length
<console>:6: error: value length is not a member of Int
     t.length
```

- #######, ######### ## #### ########### #### ######## ## #### ### ### ##########. #.#. ###### #########, ######### #####.

First-class Functions

- ##### ######### ### #####–#####, ## ### #### #### #### ## ## ## ### ### ########.
- # ######## #### ####### # ######## ## ## ######## ## # *higher-order function*.
- ### ##### ####### ######## #### ######–##### #########

```
people.filter(_.age > 20)

people.sort((p1, p2) => p1.name < p2.name)

(1 to 15).filter(even)
```

Algebraic Data Types

The Expression Problem

- ## ### ### ####### ####### ## ########### ## ## ### ###### ### ####### ## ####### ## ### # ###### ## ## ######### ### ### ### ###############.

- #### ## ## ##### ### ### #########?
- #### ## ## ##### #### ##### ## ### ### ###############!?

One or None

- ####### ## #### ## ### # ########## #### ####### ### #### #### ## ### ####### ### ## #### ## ######## ##### ## ####.

```
interface OneOrNone<T> {
  <A> A visit(OneOrNoneVisitor<A> v);
}

abstract class One<T> implements OneOrNone<T> {
  public <A> A visit(OneOrNoneVisitor<A> v) { return v.visit(this); }
  abstract T one();
}

class None<T> implements OneOrNone<T> {
  public <A> A visit(OneOrNoneVisitor<A> v) { return v.visit(this); }
}

interface OneOrNoneVisitor<A> {
  <T> A visit(One<T> o);
  <T> A visit(None<T> n);
}
```

Visitor

- ## ### ##### # ######## #### #### ### #######.

```
class You {
  static <X> String stringOne(OneOrNone<X> o) {
    return o.visit(new OneOrNoneVisitor<String>() {
      public <T> String visit(One<T> o) { return "got it! " + o.one(); }
      public <T> String visit(None<T> n) { return "don't got one"; }
    });
  }
}
```

- ## ### ########### ############ #### ### ## ### #####, ##### ### ## ##### ##### ###–##### # #####, ##### ### ##### #######.

ADTs Instead

- #######, ## ## #### #### ### ####### #### ### ###### ## *Algebraic Data Type*.

```
sealed trait OneOrNone[+A]
final case object None extends OneOrNone[Nothing]
final case class One[+A](a: A) extends OneOrNone[A]
```

- #### ## *pattern match* #### #############:

```
def stringOne[X](o: OneOrNone[X]) = o match {
  case One(o) => "get it! " + o
  case None => "don't got one"
}
```

- #### ######!

Describing structure

- ######## ### ### ## ####### #######[12].

```
sealed trait Natural
final case object Zero extends Natural
final case class Succ(s: Natural) extends Natural
```

- ## ### ### ##### # ######## ## ### ### ####### #######.

```
def add(x: Natural, y: Natural): Natural = x match {
  case Zero => y
  case Succ(t) => add(t, Succ(y))
}

scala> val k = add(Succ(Succ(Zero)), Succ(Zero)) // add 2 1
k: Natural = Succ(Succ(Succ(Zero)))
```

- ## ########### ######### #### ##### ##### *pattern matching* # ### `case` ### `match` ###–#####.
- ## ##### ##### #### ######### #### #### ######### #### ####.

List

- ##### ### ######### ###### ## ####–##### ###########.

```
sealed trait List[+A]
final case object Nil extends List[Nothing]
final case class Cons[A](h: A, t: List[A]) extends List[A]
```

- ######–###### ####, #######, ###### #### ### ### ## #### ##, ##### ### #########! ## *prepend* ## #####, ### ######[13] ### ## ## ### ## ## #############.
- ############ ##### ## #######. ### ### #### #### #### ### ####### ### ##### ### ###### ## ##### ## ###### ## # ### #### ## #######, ### ### #### ### #### ######### ###### #####.

Option

- ### `Option` ######### ## # #### #### ######## 0 ## 1 ####### (*sound familiar?*) # ### ## ### ######### #### `List`.

[12] ### ###### ####### ######## # ###### ######
[13] ## ###, ### ## ## # #### ######### #########.

```
sealed trait Option[+A]
final case object None extends Option[Nothing]
final case class Some[+A](a: A) extends Option[A]
```

- `Option` ## # ###### `null` #### `null`.
- ######## # `Map.get(K)` #### ####### ## `Option[V]` # ### #### # `V`.
- ### `flatMap` ###### ## ### ####### ## # ###### ####### #### ### `Option` ######### ####.

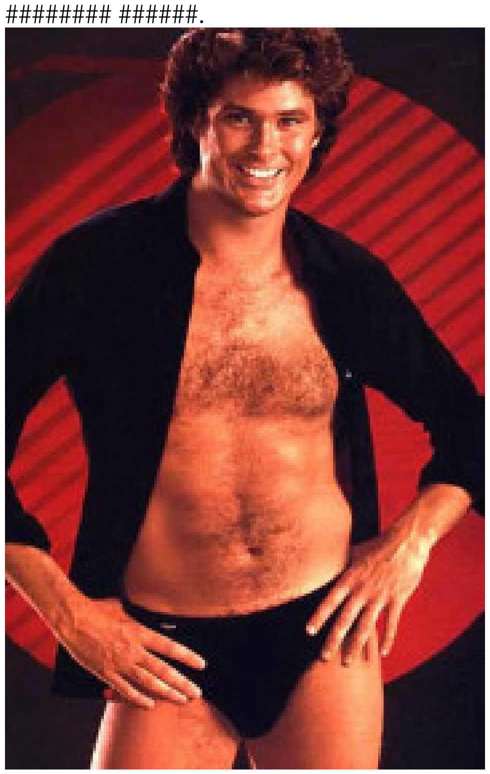
```
X x = method1(args1);
if(x == null) return null;
else {
  Y y = method2(x, args2);
  if(y == null) return null;
  else return method3(y, args3);
```

###!

```
method1(args1) flatMap (method2(_, args2)) flatMap (method3(_, args3))
```

Either

- ### `Either` ######### #### ## ########### ## #### ## ### ######## ## ## ########## ######## ######.



Either

- ### `Either` #### #### ########## *logical disjunction — this or that.*[14]
- #### ### #### ###### ## ###### ### #### ## #######, ######### ## ### #####?
- #########, ##### ##### # ##### ## ######### ### ########### ###### ## #### ########### # ####### ##### ### ####### ####### ##### ### ########### ###############.
- `Either` ## # ###### ########### ######### ##### #### #### ##########.

Others

- ##### ### #### ######; ##### (*logical conjunction*), ######### (*logical implication*), #### #####, #####.
- ### ## ######, ### ### ##### #### ###.

Partial Application

Replace Parameter With Method

########### ######## [####://###.###########.###/####–###/##########################.####]: ####### ######### #### ######

######, #### ###### ### ###### ## # ######### ### # ######. ### ######## ### #### ###### #### ######. ###### ### ######### ### ### ### ######## ###### ### ######.

######:

```
int basePrice = _quantity * _itemPrice;
discountLevel = getDiscountLevel();
double finalPrice = discountedPrice (basePrice, discountLevel);
```

```
int basePrice = _quantity * _itemPrice;
double finalPrice = discountedPrice (basePrice);
```

Java Gets No Curry

- ## ####, ### ####### ### ## *uncurried form.*
- #### #### #### ## #### ######### ### ###### # ##### #.#. `(A x B x C) -> D`.
- ## ####### ####, `A -> B -> C -> D` ##### ### -> ######### ## ### ##### `A -> (B -> (C -> D))`.
- ## ### *partially apply* #### ########, ####### #### # ######## (#### ## ### ##### ####### ######## ##) ## ## ### #### # #####.

Constructors as the Example

- ##### #### ## ###### #########, ## ##### ### ########### ######### ## #### ###### ## ## # *factory*.

[14] ### #####–###### ########## ## ###### ## ########.

- ####, ##### ### ## ### ###### ### ### ########### ######### #######.

```
Foo foo = new Foo("foo");
foo.method("bar"); // method uses the value "foo" by accessing a field
```

- ######## # ####### ####### `Encoder -> String -> byte[]`.

```
Encoder e = new Encoder("UTF-8");
byte[] encoded = e.encode(contents); // has UTF-8 applied.
```

Partially Applying Scala

- ## #####, ## ### ######### ##### ######### ## ###### ### #########.

```
scala> def add(x: Int)(y: Int) = x + y
add: (Int)(Int)Int

scala> def foo = add(7) _
foo: (Int) => Int

scala> val h = foo(8)
h: Int = 15
```

- ## ### ### #### ##### ######.

```
scala> val i = List(1, 2, 3) map foo
i: List[Int] = List(8, 9, 10)
```

- ### #######.

```
scala> def encoder: String => String => Array[Byte] = null // todo
encoder: (String) => (String) => Array[Byte]

scala> def utf8 = encoder("UTF-8")
utf8: (String) => Array[Byte]

scala> def encoded = utf8("contents")
encoded: Array[Byte]
```

Implicit Arguments

- #### ######## ######### ### ## ######## `implicit`.
- ####, ## #######-####, ### #### ##### ######## ### #### ## #### ######## ## ####. ## ##### ## ## ######## ######## ## ##### ## #### #### ###, #### # #######-#### ##### #######.
- ######### ###### #### # ######### ####### ## ########### ######, ##### #### ##### ####-#######?

- ### ########, ## ### ###### ## ######### ######## ####### ###### ####–#######[15] #### implicit #########.
- ## ### #### ## #### ### ####### ###### #### ### #####, ### #### ### ######### ## ######[16]

```
scala> implicit def i(s: String) = new { def parseInt = Integer.parseInt(s) }
i: (String)java.lang.Object{def parseInt: Int}

scala> val v = "456".parseInt
v: Int = 456
```

Real World Snippets

Instinct BDD Framework

######[17]:

```
private <T> Collection<Method> findNamedMethods(final Class<T> cls, final NamingConvention namingConvention) {
  final Collection<Method> locatedMethods = new ArrayList<Method>();
  for (final Method method : methodLocator.locate(cls)) {
    if (method.getName().matches(namingConvention.getPattern())) {
      locatedMethods.add(method);
    }
  }
  return locatedMethods;
}
```

#####:

```
def findNamedMethods[T](cls: Class[T], namingConvention: NamingConvention) =
  methodLocator locate(cls) filter (_.getName.matches(namingConvention.getPattern))
```

Reductio Website

######## #### ######## [####://############.###/########] ######### ####### ### #####–###:

```
<div>
  <ul>
    {
      examples map { case (c, name, _) =>
        <li><a href={ '#' + c }>{ name }</a></li>
      }
    }
  </ul>
  <hr/>
</div>
```

[15] ### ### ## #### ##–### ############ #### ## ### ## ###### ###### ### ####### ##### (### ########## ######).
[16] #### ### ## 3.0 ######### #######
[17] *Courtesy: Tom Adams, Instinct [http://code.google.com/p/instinct/]*

Want to Learn More?

- # #### ####### ###### #### ######### #### # ### ### # ###### # ## ##### ## #########.
- # ##### #### ## #### ##### ###### #######, ##### ## ###### ### ########## ## # ### #### ####.
- ############ ###### 2 # 3 ### #####–## ######## ######## #### ##### ##### ###### ### #### ## ######## ######. ## ###########, ## ## ###### #### ######### *why this foreign mumbo-jumbo is important*.
- ### ######## ######## ##### ######## ###### ### ### ### ## ######## ####–##### ###–######## ########, ######### ##### ##### ## #######.
- ## #### ## ###### ## #### #### ## ### ### ########## ## ######## # ###### ### ###### ########## ######## ######### ####################.### [######:####################.###].
- #### ###### #### ### ## *"aha! moment"* ## ### 2 ## 3 ### #### ####### ######### ### ##### ###### #### ##### #####.
- ### ##### ## #### #### # ##### ## ### ### ## #######?
- ##### ###### ### ######### ## ### ############ #### ####://####.############.###/.

If we have time but we probably won't

Are we still here?

- ### ######### ######## ### ##### #### ### ######## ### #### ## #### ### ######## ####### ##########.
- #### #### ## #### ## ###?

I feel very little right now

- #### ### #### #### ##### ### ###### ############ ## ############ ## ########### ####–######.
- #### ## ### #### ######## ######## ### ##### #### ### ### #### ######## ### ########## ########, #######, #### ######### ####### #### ###–######### #############.
- ##### ### #### ### ###### #### ### ######.
- #### ### ######### #### ##### ##### ### ##### #### ########### ## ### ## ### ######.

Higher Kinds

- ## ####, ## ### ######## #### #### #########. ##### ###, ## ##### ##### # `length` ####–#### ### `List<String>, List<Integer>, List<Person>` ### ## ##.

- #### #####, ## ### ######## #### *type constructors*.

```
trait Mappable[F[_]] {
  def map[A, B](fa: F[A], f: A => B): F[B]
}
```

- #### #### ############ #### ### `Mappable`:
  - `List`
  - `Option`
  - `Either[X, _]`
  - `Either[_, Y]`
  - `Function[X, _]`

Lazy Annotations

- ##### ######### ### ## ######### ## ##### ##### ##########.
- #### ### ## #### ######### ## ## ########### #### ############ ####-#######, ### #### ##### ######### ############# ##########. ## #######!
-

```
scala> def and(x: Boolean, y: => Boolean) = x && y
and: (Boolean,=> Boolean)Boolean

scala> val r = and(false, throw new Error("blow up!"))
r: Boolean = false
```

High Level Abstractions

- ##### ## #### ######-##### ## ### ##### #### ####-##### ############, ######### ##### #### ### #### ## ########### # ######## ######.
  - `Functor`
  - `Monad`[18]
  - `Applicative`[19]
  - `Traversable`[20]
  - `Arrow`[21]

Some High-level Code Snippets (for fun)

- ###### ####### (`!>` ### `unless`) ## `Boolean` ## ####### # ####-###### ######### ## ### #####.

[18] ### ###### ### ########## ########### ## ###### ######
[19] ### ########### ########### #### ####### ## ##### ####### ### #### ########
[20] ### ### ####### ## ### ######## ####### ## ###### ####### ### ##### #. #. #. ########
[21] ### ########### #### ###### ## #### ######

```
(7 > 6) !> println("seven is greater than six")
(6 > 7) unless println("six is not less than seven")
```

- ####### ####### ## ### ######### #########.

```
List(1, 3, 5) any even // false
Array(1, 2, 3) all even // false
List(2, 6, 8, 9, 6, 7, 3, 5, 8, 6, 9) selectSplit even // List(List(2, 6, 8), List(6), List(8, 6))
"age=54&name=Bob&address=At Home".toList selectSplit (_ != '&') > (_.mkString) // List(age=54, name=Bob, address=At Home)
```

Functors and Monads

- ### `Functor`.

```
List(1, 2, 3) > (_ + 1) // List(2, 3, 4)
Some(1) > (_ + 1) // Some(2)
None > (_ + 1) // None
```

- ### `Monad`.

```
Some(7) >>= (x => if(x % 2 == 0) None else Some(x + 1)) // Some(8)
Some(8) >>= (x => if(x % 2 == 0) None else Some(x + 1)) // None
Some("a").replicate[List](3) // Some(List(a, a, a))
```

- ### `MonadPlus`.

```
List(1, 2, 3) <+> List(4, 5, 6) // List(1, 2, 3, 4, 5, 6)
Some(7) <+> Some(8) // Some(7)
Some(7) <+> None // Some(7)
```

- ### `Applicative` #######.

```
Some(7) <*> (None > add2) // None
Some(7) <*> (Some(8) > add2) // Some(15)
List(1, 2, 3) <*> (List("a", "b") > (s => n => n + s + n)) // List(1a1, 2a2, 3a3, 1b1, 2b2, 3b3)
```

- ### `Traversable`.

```
Some("789") traverse ((_: String).map(_ - 48).toList) // List(Some(7), Some(8), Some(9))
List("abc", "def", "ghi") ->> // abcdefghi
Array(true, false, false, true, false) ->> // true (using the disjunction monoid)
```

Questions

#########. #########?

####################.### [######:####################.###].

A. Yeah but how can we write medium/large applications?

####–##### ########### ##### ### #### ########. #### ######## ##### ########## (## ####### ## #########), ##### ## #### ####### ########, ##### ## # ######/##### ###########?#.

######## #### #### ####### #######:

```
def f(a: Int, b: Int, c: Int) = a * b + c
```

units of software (* ### +) ## ####### # ###### (### ##### #### #### ##### ### ## #####) ##### ## ########. ## ### ## #### ####### * ### + ### *pure functions*. #### #### ## ######## ####–#######.

####–#######. #### ### ######### ## ## ############ #### ### ########### ## ## ########### ## ##### ##########. ## ####### ### #### ## ###### ### ###–######## ############ ## ########## ####–####### ###### ### ####### ####, ## ###########. ### ######## ## ## #### ## # ####### #### ##### ###########– ## ######## #### ### ########### ##### ##### ### #####. ## ###### ## * ## + ####### # ####– ######, ## #### #######, ###### ### ####–######.

#/# ####### ## # ####### ###########, ### ########## ### ###– ####### ##### ### ###############. #######, ##### ### ########### ## ## ######## #### # ###### ####### ## ## ########, ## ## ##### ######## ## ##### ## ### ########### #######– #### ### ####### # *large application* # ##### ## # ######### ### ## *unmanageable application*. ## ##### #####, ### ###### ## ### ######## ##### ## # ######/##### ###########?# ## ###### ####:

> ### ########### #### ## ## ######### ## ###### ## # ###### ## ### ######– ####### ## ####–#######, #### ## ##### ######## # ######### ## ######– ########## ## ####### # ###### #### ## ###### ######### ##### ## ##### ########### ############. ##### ############ ### ##### #### ## ###### ######### ### ########## ### ######## ## ######## ##########.

####–####### ## #########, # ## ##### #### #### #### ######## ## #### ############ ### ## ##### ## ### ###### ## ### ### ### ####### ##### ######. #######, #### #### ### ### ## ############### ######### ## ####### #####– ####### #### #### #### #### #### ### #### ## ####, ###### #### ### #### ## ############# ######### ## ### ###### ##### ## #### ##########. #### ## # #### #### ####### ### ####– ####### ## ##### ##########. ##, ######, ## ##.

(#### #######) ## # ####–#####–##–#####–###### ########### (#### #####), ##### ############# ####### ## ######## ### ######## ## ## ##### #### ######, ### ########### ####### # ####### ### ###–####### ########### #####– ##### (## #### ######).

############? #### #### ######### ### ####### ##### ## #### ### ###### #### # ###### ### (### ######). ### ##### #### ABC, ### ### ## #### B ### C? #### ###### ####, ##### A, #### ABC. #### ### *really* #####. ### #### #### ABCDEF? #### ## ### #### ABC, #### ##### D, E ### F ### ###### ###### ####. ...## #########.

######, ###### ### ############ ## #### ######, #### ######, ##### ######-#####, ###### ###### ### #### ##### ####### ### #### ####### ## ####### #### #############.

B. Scala Exercises For Beginners

```
sealed trait List[+A] {
  override def toString = {
    def toScalaList(t: List[A]): scala.List[A] = t match {
      case Empty => Nil
      case Cons(h, t) => h :: toScalaList(t)
    }
    toScalaList(this).toString
  }
}
final case object Empty extends List[Nothing]
final case class Cons[A](h: A, t: List[A]) extends List[A]

object List {
  def foldRight[A, B](as: List[A], b: B, f: (A, B) => B): B = as match {
    case Empty => b
    case Cons(h, t) => f(h, foldRight(t, b, f))
  }

  def foldLeft[A, B](as: List[A], b: B, f: (B, A) => B): B = as match {
    case Empty => b
    case Cons(h, t) => foldLeft(t, f(b, h), f)
  }

  def reduceRight[A](as: List[A], f: (A, A) => A): A = as match {
    case Empty => error("bzzt. reduceRight on empty list")
    case Cons(h, t) => foldRight(t, h, f)
  }

  def reduceLeft[A](as: List[A], f: (A, A) => A): A = as match {
    case Empty => error("bzzt. reduceLeft on empty list")
    case Cons(h, t) => foldLeft(t, h, f)
  }

  def unfold[A, B](b: B, f: B => Option[(A, B)]): List[A] = f(b) match {
    case Some((a, b)) => Cons(a, unfold(b, f))
    case scala.None => Empty
  }
}

sealed trait Natural {
  override def toString = {
    def toInt(n: Natural): Int = n match {
      case Zero => 0
      case Succ(x) => 1 + toInt(x)
    }
    toInt(this).toString
  }
}
final case object Zero extends Natural
final case class Succ(c: Natural) extends Natural
```

```scala
object Exercises {

// Exercise 1
// Relative Difficulty: 1
// Correctness: 2.0 marks
// Performance: 0.5 mark
// Elegance: 0.5 marks
// Total: 3
def add(x: Natural, y: Natural): Natural = error("todo")

// Exercise 2
// Relative Difficulty: 2
// Correctness: 2.5 marks
// Performance: 1 mark
// Elegance: 0.5 marks
// Total: 4
def sum(is: List[Int]): Int = error("todo")

// Exercise 3
// Relative Difficulty: 2
// Correctness: 2.5 marks
// Performance: 1 mark
// Elegance: 0.5 marks
// Total: 4
def length[A](as: List[A]): Int = error("todo")

// Exercise 4
// Relative Difficulty: 5
// Correctness: 4.5 marks
// Performance: 1.0 mark
// Elegance: 1.5 marks
// Total: 7
def map[A, B](as: List[A], f: A => B): List[B] = error("todo")

// Exercise 5
// Relative Difficulty: 5
// Correctness: 4.5 marks
// Performance: 1.5 marks
// Elegance: 1 mark
// Total: 7
def filter[A](as: List[A], f: A => Boolean): List[A] = error("todo")

// Exercise 6
// Relative Difficulty: 5
// Correctness: 4.5 marks
// Performance: 1.5 marks
// Elegance: 1 mark
// Total: 7
def append[A](x: List[A], y: List[A]): List[A] = error("todo")

// Exercise 7
// Relative Difficulty: 5
// Correctness: 4.5 marks
// Performance: 1.5 marks
// Elegance: 1 mark
// Total: 7
def flatten[A](as: List[List[A]]): List[A] = error("todo")
```

```
// Exercise 8
// Relative Difficulty: 7
// Correctness: 5.0 marks
// Performance: 1.5 marks
// Elegance: 1.5 mark
// Total: 8
def flatMap[A, B](as: List[A], f: A => List[B]): List[B] = error("todo")

// Exercise 9
// Relative Difficulty: 8
// Correctness: 3.5 marks
// Performance: 3.0 marks
// Elegance: 2.5 marks
// Total: 9
def maximum(is: List[Int]): Int = error("todo")

// Exercise 10
// Relative Difficulty: 10
// Correctness: 5.0 marks
// Performance: 2.5 marks
// Elegance: 2.5 marks
// Total: 10
def reverse[A](as: List[A]): List[A] = error("todo")
}
```