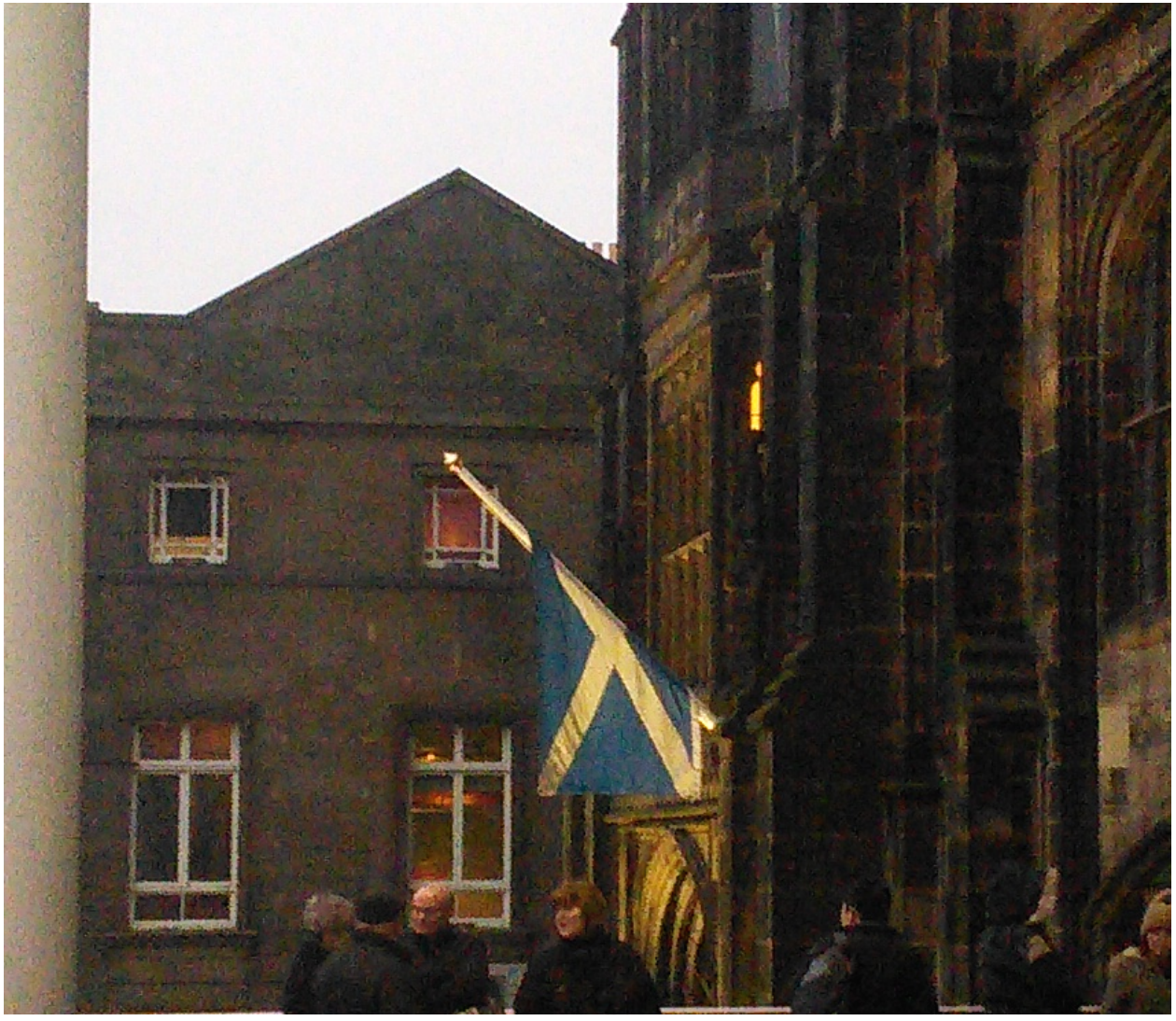
Embedded Domains Specific Languages in Idris
Lecture 1: Fundamentals of Idris

Edwin Brady (ecb10@st-andrews.ac.uk)
University of St Andrews, Scotland, UK
@edwinbrady

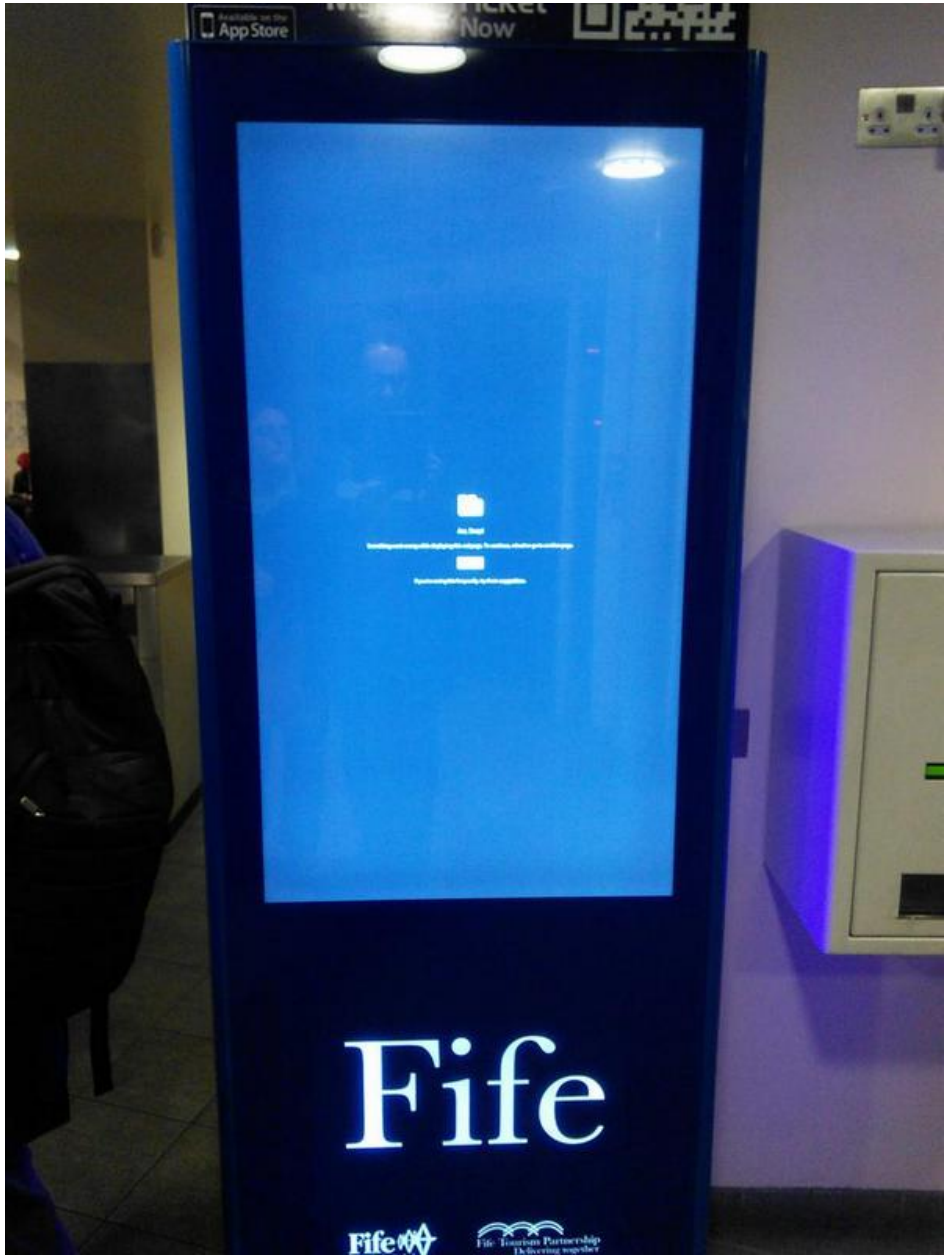
SSGEP, Oxford, 6th July 2015

sicsa*

Scotland



Welcome to Fife



Security Error!

IDRIS is a *Pac-man Complete* functional programming language with *dependent types*

- `cabal update; cabal install idris`
- `http://idris-lang.org/download`
- `http://idris-lang.org/SSGEP/`

sicsa*

Idris is a *Pac-man Complete* functional programming language with *dependent types*

- `cabal update; cabal install idris`
- `http://idris-lang.org/download`
- `http://idris-lang.org/SSGEP/`

This course is in three parts:

- *Today:* Fundamentals: A tour of Idris
- *Tomorrow:* Writing Domain Specific Languages in Idris
- *Thursday:* Managing *side-effects* and *resources*

sicsa*

Why types?

We can use *type systems* for:

- *Checking* a program has the intended properties
- *Guiding* a programmer towards a correct program
- Building *expressive* and *generic* libraries

Why types?

We can use *type systems* for:

- *Checking* a program has the intended properties
- *Guiding* a programmer towards a correct program
- Building *expressive* and *generic* libraries

Dependent Types allow types to be predicated on *values*

- More precise specification of properties
- More help from the compiler
- Types as a *first class* language construct

- Full dependent types
  - Functions can compute types, types can contain values

- Full dependent types
  - Functions can compute types, types can contain values
- Compiled (via C, Javascript, . . .)
  - Some optimisations, e.g. aggressive erasure, inlining, partial evaluation

- Full dependent types
  - Functions can compute types, types can contain values
- Compiled (via C, Javascript, . . .)
  - Some optimisations, e.g. aggressive erasure, inlining, partial evaluation
- Type classes, like Haskell
  - No `deriving`, yet
  - `Functor`, `Applicative`, `Monad`, `do` notation, idiom brackets

- Full dependent types
  - Functions can compute types, types can contain values
- Compiled (via C, Javascript, . . .)
  - Some optimisations, e.g. aggressive erasure, inlining, partial evaluation
- Type classes, like Haskell
  - No `deriving`, yet
  - `Functor`, `Applicative`, `Monad`, `do` notation, idiom brackets
- *Strict* evaluation order, unlike Haskell
  - `Lazy` as a type

- Full dependent types
  - Functions can compute types, types can contain values
- Compiled (via C, Javascript, . . .)
  - Some optimisations, e.g. aggressive erasure, inlining, partial evaluation
- Type classes, like Haskell
  - No `deriving`, yet
  - `Functor`, `Applicative`, `Monad`, `do` notation, idiom brackets
- *Strict* evaluation order, unlike Haskell
  - `Lazy` as a type
- Codata (badly)

- Full dependent types
  - Functions can compute types, types can contain values
- Compiled (via C, Javascript, . . .)
  - Some optimisations, e.g. aggressive erasure, inlining, partial evaluation
- Type classes, like Haskell
  - No `deriving`, yet
  - `Functor`, `Applicative`, `Monad`, `do` notation, idiom brackets
- *Strict* evaluation order, unlike Haskell
  - `Lazy` as a type
- Codata (badly)
- Foreign functions, system interaction