"Pascal [Java] is for building pyramids – imposing, breathtaking, static structures built by armies pushing heavy blocks into place. Lisp [Haskell] is for building organisms – imposing, breathtaking, dynamic structures built by squads fitting fluctuating myriads of simpler organisms into place.

...

the pyramid must stand unchanged for a millennium; the organism must evolve or perish."

– *Alan J. Perlis, Foreword to "Structure and Interpretation of Computer Programs", 1985, 1996*

The plan going forward

- This week: input-output, algebraic data types
- Assignment 3 due next week provides some templates for possible projects (algebraic types, reading files, user interaction)
- Project (groups of 2 or 3):
  - Proposal: 12 February
  - Final Project Due: 26 February
  - Project Demos: 27 Feb - March 4 (by appointment with TAs)
- In class: worked examples (games in Haskell)
- Midterm 2 on 26 February
- Then Logic Programming!!

Some Representative Survey Feedback

- I dislike:
  How much I need to google to complete the assignments
  Up until now, the course focus a lot on the theory side
  Trying to understand how to code by watching someone else
  We go through the iclicker answers very quickly.
  Topics build on each ther very fast and it feels like sometimes there isn't enough time to solidify the groundwork.
  Nothing, I beleive the way the course is taught right now is perfect and is helping me learn a lot
  It sometimes is too slow.
  The pace of the lectures, seems a bit quick sometimes
- I wish:
  to learn more applications of Haskell or any functional in real world settings (not only theory).
  I wish the midterm is really easy and I do well in this course :)
  I wish the answers to the clickers on the slides are posted somewhere

Input-Output (see IOAdder.hs)

- `IO t` is a type that does input output
- Output `putChar`, `putStr`, `putStrLn`
- Input `getChar`, `getLine`
- Do:

```
do v1 <- a1
   v2 <- a2
   ...
   vn <- an
   return (f v1 v2 ... vn)
```

  Evaluates each ai in turn, save value in vi
  `vi <-` is optional if you don't want to save
  returns value of `(f v1 v2 ... vn)`

- do could end with calling another function that has a return

Input-Output (see IOAdder2.hs)

- `IO t` is a type for input and output
- `type IO t = World -> (t,World)`
  where `World` contains information about state of world.
- ```
  do v1 <- a1
     ...
     vn <- an
     return (f v1 ... vn)
  ```
  Each *ai* is of type `IO ti` for some type *ti*
  vi is of type *ti*
  ai gets world from a_{i-1}, gives value to v_i and world to a_{i+1}
- Type of do expression is `IO t` where t is return type of f
- To define a new value use
  `let v = exp`
  where type of `v` is type of `exp`
  ```

Return

- `IO t` is a type for input and output
- `type IO t = World -> (t,World)`
  where `World` contains information about state of world.
- ```
  do v1 <- a1
     ...
     vn <- an
     return (f v1 ... vn)
  ```
  returns `(v,world)` where v is value of `(f v1 ... vn)`
- How is return defined?

  ```
  return v world = (v,world)
  ```

  which is returned as the value for *do*.
  Value of v is printed in interactive mode.
- IO is a *monad*.
  ```

Clicker Question

Consider the program:

```
foo =
  do
    putStrLn("Test in foo")
    return (3 :: Integer)
```

What is the type of foo?

A `foo :: [Char]`

B `foo :: IO [Char]`

C `foo :: Integer`

D `foo :: IO Integer`

See TestDo.hs

Clicker Question

Consider the program:

```
foo =
  do
    putStrLn("Test in foo")
    return 3
```

What output from evaluating foo in ghci?

A
```
Test in foo
3
```

B
```
3
Test in foo
```

C
```
3
```

D
```
"Test in foo"
```

Clicker Question

```
foo =
  do
    putStrLn("Test in foo")
    return (3 :: Integer)
bar =
   do
      putStrLn("Test in bar")
      v <- foo
      putStrLn ("v is "++show v)
      return ("v^3 is "++show (v^3))
```

What is the inferred type of bar?

A `bar :: [Char]`

B `bar :: IO [Char]`

C `bar :: Integer`

D `bar :: IO Integer`

See TestDo.hs

Clicker Question

```
foo = do
        putStrLn("Test in foo")
        return 3
bar = do
        putStrLn("Test in bar")
        v <- foo
        w <- v+7      ---this line---
        putStrLn ("v is "++show v)
        return ("v^3 is "++show (v^3))
```

This given an error with `---this line---` included, but not with it removed. Why?

A v is not a number and so cannot be added to 7

B `v+7` is a Num type, not of type IO t

C w is not used in the rest of the definition

D `---this line---` is illegal at the end of the line

Clicker Question

```
foo = do
        putStrLn("Test in foo")
        return 3
bar2 = do
        putStrLn("Test in bar")
        v <- foo
        w <- 7
        putStrLn ("v is "++show v)
        return ("v^3 is "++show (v^3))
```

What error message does Haskell produce

A No instance for `(Show a0)` arising from a use of `'print'`
B Runtime error: `'7'` is not an `IO t0`
C You are not allowed to have `'w <- 7' in a 'do'`
D No instance for (Num (IO t0)) arising from the literal `'7'`
E parse error (possibly incorrect indentation or mismatched brackets)

Clicker Question

```
foo = do
        putStrLn("Test in foo")
        return 3
bar3 = do
        putStrLn("Test in bar")
        v <- foo
        w <- v
        return ("v^2 is "++show (v^2))
```

Why does Haskell produce the error:

`No instance for (Num (IO t0)) arising from the literal '3'`

A It is a typo; it should say `w <- v` is wrong

B It is possible that `w <- v` is legal expression if v is of type `(IO t0)` but v must also be in the class `Num`

C The error messages are designed to be confusing

D "return 3" is illegal in foo

Clicker Question

```
afun :: IO Int
afun =
  do
    aaa <- return 5
    return (aaa+4)
```

What does `ghci` print when `afun` is called:

A 5

B 9

C 5
  9

D It gives a compilation error.

E It gives a run time error.