What's the difference between `Int` and `Integer`?

`Int` is fixed width, its exact size depending on the hardware. It silently overflows.

`Integer` is arbitrary sized.

Why is `Float` discouraged?

Haskell compilers writers focus their effort on making `Double` efficient.

What's a type signature?

The combination of `::` and the type after it.

What is Haskell's syntax for function calls?

The function name, a space, and then the arguments separated by spaces too.

```
compare 3 2
```

What's a composite data type?

A type constructed from other types, like lists and tuples.

What is a type variable in Haskell?

A variable that ranges over types instead of values.

What are the capitalization rules for variables in Haskell?

Type variables must start with a lowercase letter, and type names with an uppercase letter.

Describe "unit" and 1-tuples in Haskell.

Both the type name and it's sole value are written `()`.

There are no 1-tuples.

Describe `take` and `drop`.

Given a number `n` and a list, `take` returns the first `n` elements of the list, while `drop` returns all *but* the first `n` elements of the list.

How do you access the elements of a pair?

`fst` and `snd`

Describes function association.

It is the tightest association, and runs left to right.

```
a b c d
```

is equivalent to

```
(((a b) c) d)
```

How do you split a string into a list of strings?

```
ghci> lines "the quick\nbrown fox\njumps"
["the quick","brown fox","jumps"]
```

How can you tell if a function has side effects?

Type type of the function's result will begin with `IO`:

```
ghci> :type readFile
readFile :: FilePath -> IO String
```

How is Haskell's `if` different from the one in C-like languages?

- `if` expressions result in a value.
- The parens around the condition are inferred.
- There's a `then` keyword to indicate the first branch.

How is indentation significant in Haskell?

It indicates that a line *continues* an indentation instead of starting a new one.

What is `null`?

A function that indicates whether a list is empty.

What does lazy evaluation entail?

- Delayed evaluation: evaluating an expression only when the result is required by a calling function.
- Short-circuit evaluation: evaluating an expression only to the extent it is required by the calling function.
- Applicative-order evaluation: evaluating an expression at most once.

What's the difference between `putStrLn` and `print`?

`putStrLn` only takes `String` arguments, `print` takes anything instance of `Show`.

Compare how short-circuiting `||` is accomplished in Haskell and in Scala.

In Scala short-circuiting is a result of by-name parameters, a way of achieving the standard Haskell non-strict evaluation. By-name parameters do not adhere to applicative-order evaluation. Haskell doesn't do anything special, expressions always short-circuit.

Make a list of all natural numbers.

[1..]

What is the type signature of `last`?

How is it read?

`

last :: [a] -> a
`

"takes a list, all of whose elements have some type `a`, and returns a value of the same type `a`"

What is a polymorphic function?

What kind of polymorphism does it reflect?

One that has type variables in its signature, indicating that some of its arguments can be of any type.

This is an example of parametric polymorphism.

What kind of polymorphism is provided by subclass mechanisms?

Does Haskell have it?

Subtype polymorphism. Haskell isn't object-oriented so it doesn't offer this polymorphism.

What kind of polymorphism is provided by implicit conversion?

Does Haskell have it?

Coercion polymorphism. Haskell avoids it in even the simplest of cases. *E.g.*, integers are not converted to floating point numbers.

What is the signature of `take`?

```
ghci> :t take
take :: Int -> [a] -> [a]

Int -> ([a] -> [a])
```