Concatenate strings.

```
str1 ++ str2
```

What are strings, really?

Lists of characters.

How is negation done?

`/=` and unary `not`

Give the conditional syntax.

What is unusual about Haskell conditionals?

```
if bool then expr1 else expr2
```

The else clause is required, to ensure an interesting result for the whole `if` expression.

Like in Java, Boolean types are ...

... strict.

Find the type of an expression

Make `ghci` print all types.

`:t expr` shows a value's type in `ghci`.

`:set +t` makes it permanent for that session.

What are the two types of a function?

The optional type declaration and the function declaration.

Create a module `MyMath` with the function `my_max`.

```
module MyMath where
    my_max :: Integer -> Integer -> Integer
    my_max x y = if x > y then x else y
```

How can you control flow of control to a function?

- Multiple function definitions using pattern-matching parameters.
- Guards.

Give the syntax of guards.

What is often used as the last guard?

Delay the equals sign, splitting up the argument lists.

```
func args
    | boolean1 = ...
    | boolean2 = ...
```

`otherwise` is an alias for true, often used as the last guard.

Give the syntax of tuples and lists.

Tuples use round parens, lists use brackets.

Compose functions

You can do it explicitly with parens and parameters, or with the dot notation:

```
composedFunc arg = secondToApply (firstToApply arg)
composedFunc = secondToApply . firstToApply
```

Access the length of a collection.

```
length coll
```

Give the list range syntax.

```
[start ..]                --infinite list
[start .. end]            --default increment of 1
[start, second .. end] --uses second to show increment
```

Give the list comprehension sytax.

```
[expr | genOrFilter1, ..., genOrFilterN]
```

Filters are boolean expressions.

Generator form:

```
bindingForm <- collection
```

Give the anonymous function syntax.

```
\param1 ... paramN -> body
```

How can you bind variables with maximally limited scope?

You can append an indented `where` clause.

How do you partially apply a function?

Just don't give a function all its arguments. No special syntax is required.

What's the simplest way to create a new type?

```
data NewType = Val1 | Val2 | ... | ValN
```

What are some simple type alias forms?

```
type NewType = OldType
type NewType = (Type1, ..., TypeN)
type NewType = [SomeType]
```

How can you make your custom type printable?

Add `deriving (Show)` to the end of the constructor.
k

How are variables interpreted in a function's type declaration?

They are interpreted as type variables.

What is a class in Haskell?

A collection of function signatures that any instance of the class (a **type** not an object) must support.

How can you temporary variables for use in a function?

```
let var1 = expr1
    ...
    var2 = expr2
in result-expr
```

Fundamentall, monads are ...

Some common uses are ...

... a way of composing functions.

... sequential execution, control structures, managing i/o, the `Maybe` monad.