

What are Haskell's case rules?

Variables begin with a lowercase letter.

Type names begin with an uppercase letter.

Give the comment syntax.

-- comment out the rest of the line.

```
{- multiline comment -}
```

(nesting is allowed)

Describe Haskell's Boolean types?

True and False are of type Bool.

The type is strict, only admitting those two literals.

What are the Boolean operators?

Equality/Inequality?

&&, ||, not

'==' and '/='

Haskell's math operators have types that are
...

... strict, since there's no coercion. They have integer and floating point versions.

Convert `Float` to `Int`.

truncate

Convert a `Char` to an `Int` and back.

```
ord :: Char -> Int  
chr :: Int -> Char
```

What is a string, really?

A list of `Chars`, that is, `[Char]`.

Get string representations of objects.

show whatever

Make a prefix operator infix.

Make an infix operator prefix.

Surrounding an identifier with back ticks makes it infix.

Surrounding an operator with parens makes it prefix.

Index a list.

Test for membership.

```
ghci> ['a', 'b', 'c'] !! 0  
'a'
```

elem, notElem

Remove duplicates in a list.

Concatenate two lists.

nub

concat

Access the elements of a 2-tuple.

fst, snd

What is the range syntax?

$[a..b]$ is a list of all the values from a to b , inclusive.

$[a..]$ is an infinite list from a up.

What is the syntax of list comprehensions?

[expr | generatorOrGuard1, ... ,generatorOrGuardN]

Guards are just expressions that result in `Bool`. No `if` is used.

How are lambdas written?

\params -> result

How are new variables introduced?

expression where declarations

let declarations in expressions

The expression can have multiple new variables in it.

How are `case` expressions introduced?

```
case expr1 of  
  expr2 -> ...  
  expr3 -> ...
```

How are function guards introduced?

```
func params =  
  | boolean1 -> ...  
  | boolean2 -> ...
```

What is a lazy data structure?

What's an easy way to make them in Clojure?

A data structure whose parts don't exist until they are accessed.

`iterate` takes a function f and a starting value n and produces a lazy infinite series:

$(n, f(n), f(f(n)), f(f(f(n))), \dots)$

What does the `&` mean in a function signature?

It comes before a vararg, which is available as a list in the body.

Describe `do`.

`(do exprs*)`

Evaluates the expressions in order and returns the value of the last.

Describe list comprehension syntax.

`(for seq-exprs body-expr)`

Takes a vector of one or more binding-form/collection-expr pairs, each followed by zero or more modifiers, and yields a lazy sequence of evaluations of expr.

Supported modifiers are: `:let` [binding-form expr ...], `:while` test, `:when` test.

How do you partially apply functions?

Use `partial`, followed by a function and fewer than the normal number of arguments.

```
user=> (def equals5 (partial = 5))  
#'user/equals5  
user=> (equals5 5)  
true
```


Access the elements of a tuple.

fst **and** snd