How do function signatures associate with respect to partial application?

They associate to the right.

```
a -> a -> a
```

is just the same as

```
a -> (a -> a)
```

If a type parameter uses both arithmetic and ordering it must ...

... be a member of both `Num` and `Ord` since `Num` is not a type of `Ord`.

Partially apply an infix function.

Use sections:
Surround the infix function with parens and only supply an argument on one side but not the other.

What's the problem-case of sections?

How can you get around it?

`-` means negative before it means minus.

Use `subtract` instead of `-`.

What should you do if you're unsure of the type of a function you're writing?

See what Haskell infers using `:t`.

Create a new function that reverses the order of the input function's first two parameters.

```
ghci> :t flip
flip :: (a -> b -> c) -> b -> a -> c
```

What's annoying about `length`?

It returns an `Int` instead of a `Num a`.

What happens if you don't put parens around a lambda?

The function body will extend to the end of the line.

How is pattern matching params different in lambdas?

Only one pattern is allowed.

What are `scanl`/`scanr`?

They are like `foldl`/`foldr`, but they report all intermediate accumulator states as a list.

`scanl` places the final result last, `scanr` puts it first.

What are `foldl1`/`scanl1`/etc?

They do not require explicit starting values. They assume the first (or last, depending on direction) value as the starting value.

Define `$`.

What's its point?

```
($) :: (a -> b) -> a -> b
f $ x = f x
```

It's function application with the lowest (as opposed to the highest) precedence. It is typically used to introduce right association to minimize the number of needed parens.

Define ..

```haskell
(.) :: (b -> c) -> (a -> b) -> a -> c
f . g = \x -> f (g x)
```

In general, the direction of association ...

... cannot be determined from the name of a function alone.

How can composition be used with functions of several parameters?

The functions must be partially applied until each function just takes one function.

What's a common use of composition?

Using partial application in a function whose result is the argument to another function.

```
fn x = ceiling (negate (tan (cos (max 50 x))))
fn = ceiling . negate . tan . cos . max 50
```

What is a common overuse of lambdas?

Using a lambda instead of a partially applied function when passing a function to a higher-order method.

If the only purpose of the lambda is to introduce a parameter to a function, you probably could have left it off and used partial application instead.