Quiz

Ivan Trepakov

NSU Sys.Pro

Quiz 1

Evaluate expression

```
succ (2 - pred 1)

drop 2 (take 4 "Haskell") > map succ "cat"

sum (map fromEnum (enumFrom False))

map (\x -> x * 2) [1,3..10] ++ [100,1000]
```

Guess type signature

```
max "Haskell"

map fst

map (take 2)
```

Quiz 2

Guess type signature

```
(++) [True]
```

```
zip [0..]
```

```
map filter
```

Guess type signature

```
alph = 'a' : alph
```

```
foo = zipWith (:)
```

```
f = f f
```

```
y g = g (y g)
```

Quiz 3

Guess type signature

```
z x y = zip x (concat y)

concatMap f x = concat (map f x)

f = 0 : 1 : zipWith (+) f (tail f)
```

Guess the function(s)

```
_ :: a -> a

_ :: a -> b

_ :: a -> [a] -> [a]

_ :: [a] -> Maybe (a, [a])
```

Quiz 4

Guess the function(s)

```
_ :: a -> b -> a

_ :: (a -> b -> c) -> b -> a -> c

_ :: ((a, b) -> c) -> a -> b -> c

_ :: (a -> b -> c) -> (a, b) -> c
```

Guess the function(s)

```
_ :: (a -> b) -> a -> b

_ :: (b -> c) -> (a -> b) -> a -> c

_ :: (b -> a -> b) -> b -> [a] -> b

_ :: (a -> b -> b) -> b -> [a] -> b
```

Quiz 5

Evaluate expression

```
(2^) . (3+) $ 4
```

```
map ($2) [(*2), (^3), (1+)]
```

```
(++ "!") . reverse $ "abc"
```

Guess type signature

```
flip const
```

```
const undefined
```

```
foldr (:) []
```

```
((filter even .) .)
```

Quiz 6

Guess type signature

```
filter (const True)

foldl' (flip (:)) []

map (,)
```

How many distinguishable *total* functions?

```
_ :: a -> a

_ :: (a, a) -> (a, a)

_ :: a -> a -> Bool

_ :: Eq a => a -> a -> Bool
```

Quiz 7

How many distinguishable *total* functions?

```
_ :: a -> b -> a
```

```
_ :: a -> a -> a
```

```
_ :: (b -> c) -> (a -> b) -> (a -> c)
```

How many distinguishable *total* functions?

```
_ :: a -> Maybe a
```

```
_ :: a -> Maybe b
```

```
_ :: (a -> Maybe b) -> [a] -> [b]
```

Quiz 8

How many distinguishable *total* functions?

```
_ :: Bool -> Bool -> Bool

_ :: Bool -> a -> a

_ :: [a] -> a
```

Guess the function(s)

```
_ :: (a -> a -> Bool) -> [a] -> [[a]]

_ :: (a -> a -> Ordering) -> [a] -> [a]

_ :: Ord a => (b -> a) -> b -> b -> Ordering
```

Q&A