PROGRAMMING IN HASKELL

Chapter 7.1 – Before we look at Stack

A few things before we start

- map
- IO in Haskell
- Data.Text

We will revisit these in more detail in the coming weeks

The map Function

The higher-order library function called map applies a function to every element of a list.

```
map :: (a → b) → [a] → [b]
```

For example:

```
> map (+1) [1,3,5,7]

[2,4,6,8]
```

The map Function

We will examine more higher-order functions later

```
map :: (a → b) → [a] → [b]
```

For example:

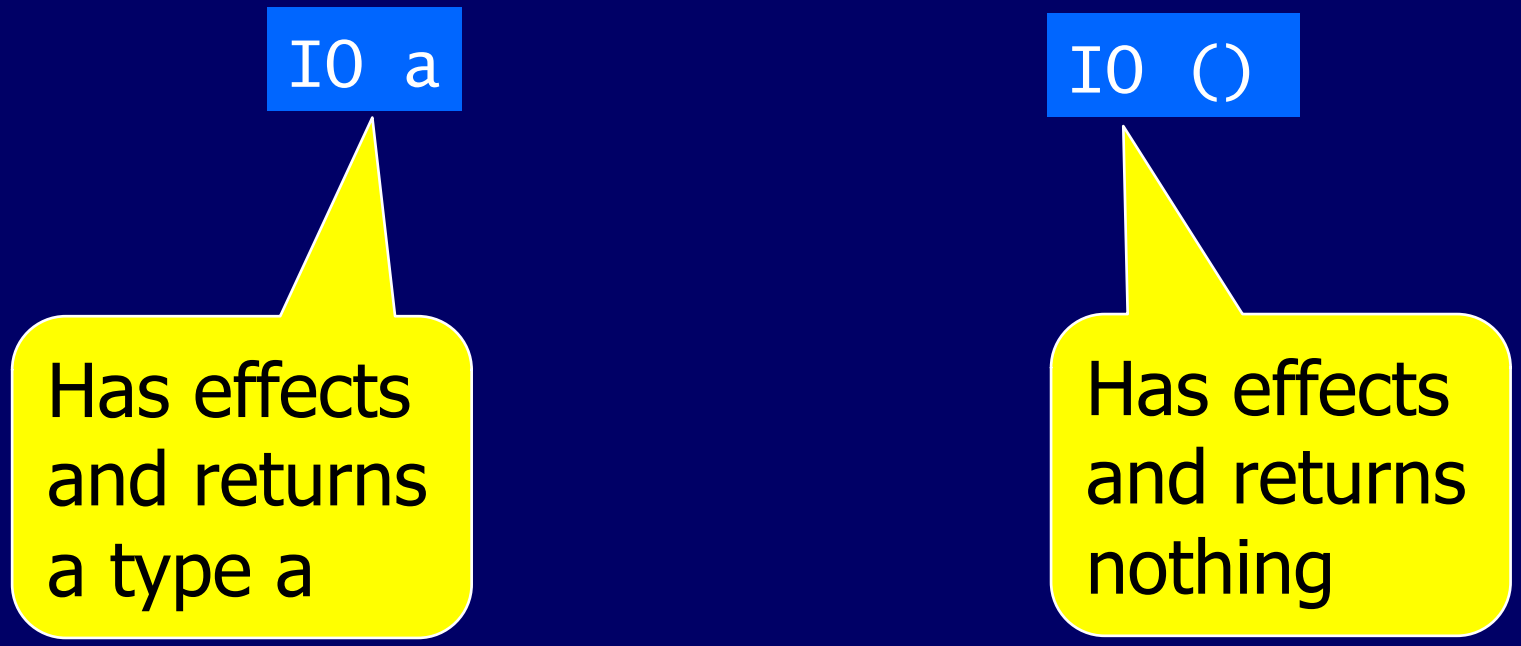
```
> map toUpper ”cat”
>”CAT”
```

IO Briefly

- Purity in Haskell means that functions should not have any side-effects.
- This means that functions should not effect state or change the outside world in any way
- This includes the use of IO (e.g., reading from keyboard, writing to console)
- We need to use such IO but the Haskell compromise is that anything with such side-effects is clearly marked as such.. How..

IO Briefly

- We type the function to clearly show that IO is involved



IO Briefly – the main function

- The Haskell compiler looks for a special value

```
main :: IO ()
```

- This will actually get handed to the runtime system and executed.

Data.Text

- This is strongly preferred over String for real-world text

- As it clashes with Prelude for a number of functions, best to use with qualified, i.e.

```
import qualified Data.Text as T
```

Note when using you need to add the following at the top of the file :

```
{-# LANGUAGE OverloadedStrings #-}
```

Data.Text

We will use Data.Text when working on some kinds of data:

- See more at

https://hackage.haskell.org/package/text-1.2.4.1/docs/Data-Text.html

See lab exercise on Data.Text for an example on using Data.Text and some of its functions:

toLower,
filter,
pack, unpack


ANY
QUESTIONS?