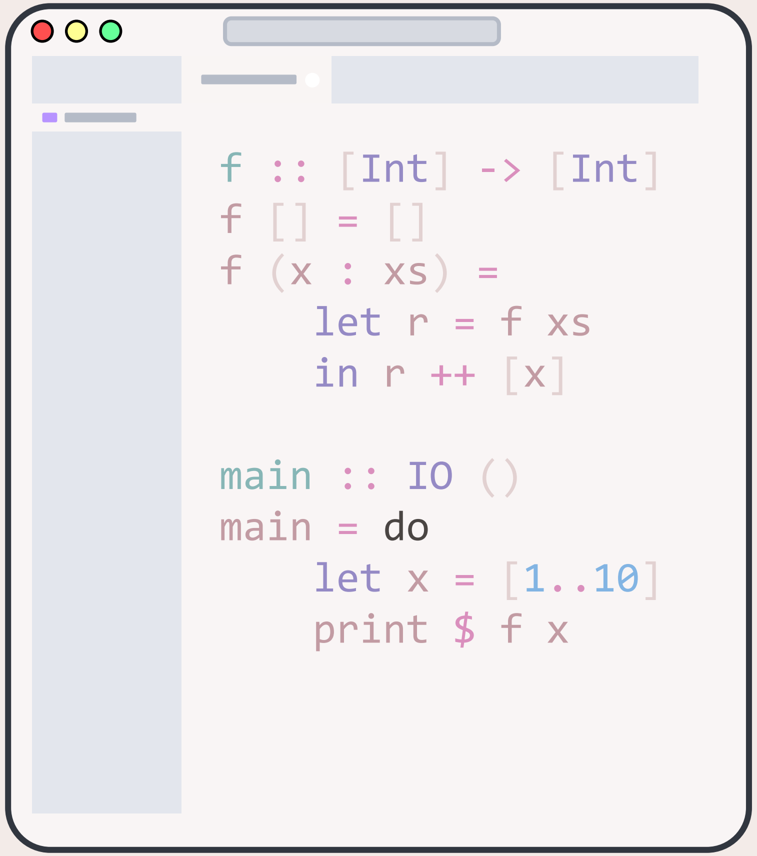
IT5100A

Industry Readiness:
Typed Functional Programming

Course Conclusion

Foo Yong Qi

yongqi@nus.edu.sg



```
f :: [Int] -> [Int]
f [] = []
f (x : xs) =
  let r = f xs
  in r ++ [x]

main :: IO ()
main = do
  let x = [1..10]
  print $ f x
```

Outstanding Administrivia

- Assignment 3 due 17 Nov 23:59, no extensions (it is the end of the semester!)
- Assignment 2 + 3 + PE grades to be released after we are done marking
- Course feedback/review will be helpful:
  - First iteration of fresh revamp from previous iteration
  - My first time running this course

Recap: Programming Paradigms

Imperative

Procedural

Programs as series of **procedures**

Object-Oriented

Objects with data and behaviour

Declarative

Logic

Programs as sets of logical statements

Functional

Programs as composition of functions

Recap: FP Principles

- Immutability
- Pure Functions
- Recursion
- Types
- First-Class Functions

Immutability

Only use immutable data

```python
# Python
def add_one(fraction):
    """fraction is a tuple of (num, den)"""
    old_num, den = fraction
    num = old_num + den
    return (num, den)

my_fraction = (3, 2)
new_fraction = add_one(my_fraction)

print(new_fraction) # (5, 2)
print(my_fraction) # (3, 2)
```

Pure Functions

Pure functions only receive input and return output

They do not **produce side effects** or **depend on external state**

Recursion

Recursive functions simulate loops

Types

Adhering strictly to type information **eliminates type-related bugs** and makes functions **transparent**

Adherence to type information can be **automatically verified by a program**

First-Class Functions

Functions are **objects**; higher-order functions support **code-reuse**

Practical Takeaways

- Prefer immutability, type safety, pure functions
- Recursion and higher-order functions can hinder performance but great for quick prototyping

Course Introduction

- Course Administration
- Functional Programming
- Introduction to Haskell

Types

Typeclasses

Railway Pattern

Monads

Concurrent Programming

Course Conclusion

Practical Takeaways

- Write type annotations for documentation and type checking
- Use type checkers like pyright to check if your types are correct
- Let types guide your programming
- Match against value/structure of data

Course Introduction

Types

- Types and Type Systems
- Polymorphism
- Algebraic Data Types
- Pattern Matching

Typeclasses

Railway Pattern

Monads

Concurrent Programming

Course Conclusion

Practical Takeaways

- Decouple data and behaviour for extensibility

Course Introduction

Types

Typeclasses

- Ad-Hoc Polymorphism
- Typeclasses
- Commonly-Used Typeclasses
- Functional Dependencies
- The Existential Typeclass “Antipattern”

Railway Pattern

Monads

Concurrent Programming

Course Conclusion

Practical Takeaways

- FP principles imply transparent functions
- Use data structures like Maybe, Either etc. for transparent function effects
- Use methods like map and flatMap to easily perform operations on Functors, Monads etc.

Course Introduction

Types

Typeclasses

Railway Pattern

- Functors
- Applicative Functors
- Validation
- Monads

Monads

Concurrent Programming

Course Conclusion

Practical Takeaways

- Monads are everywhere, many data structures/libraries are monads
- Monads have similar patterns (reading/writing state), now you know how to use them better

Course Introduction

Types

Typeclasses

Railway Pattern

Monads

- More about Monads
- Commonly-Used Monads
- Monad Transformers

Concurrent Programming

Course Conclusion

Practical Takeaways

- Mutability often inevitable in concurrency/parallelism
- FP has ideas like STM for “safely” mutating structures by performing atomic transactions

Course Introduction

Types

Typeclasses

Railway Pattern

Monads

Concurrent Programming

- Concurrent Programming with Threads
- Parallel Programming
- Software Transactional Memory

Course Conclusion

Beyond IT5100A

- Learning FP in Haskell opens you up to FP used in industry:
  - OCaml (finance, e.g. Jane Street)
  - Scala (data e.g. GovTech, web applications e.g. Lichess, LinkedIn)
  - Haskell (e.g. Google, Twitter, IBM, NVIDIA, Microsoft, Tesla, finance e.g. Standard Chartered)
- Now you are more familiar with declarative styles of programming (remember, monads are everywhere!)
- Programming languages research (if you're interested)

Beyond IT5100A

You can show off to your friends/potential employers that you know Haskell!

Thank you

Hope you had a great 7 weeks!

Foo Yong Qi

yongqi@nus.edu.sg

https://yongqi.foo/