Title

Subtitle

Timothy Ballet
Department of blank
University of blank

WARWICK
THE UNIVERSITY OF WARWICK

1 Introduction

Write around four paragraphs establishing the context and motivating your project.

1.1 Related work

Discuss related work.

```
    int testy
```

1.2 Objectives

One sentence summary of your project. Followed by a short list of concrete objectives:

- Objective 1
- Objective 2
- Objective 3

2 Timeline

This module is comprised of approximately 30 lectures, 10 labs, 2 pieces of coursework, and an exam. This section contains a chronological schedule of all of these components. Note that the schedule may be subject to changes due to *e.g.* staff illness or other unforeseen circumstances. Each lecture aims to answer a specific question, which is shown in the timeline. You can test your understanding by asking yourself that question after each lecture and checking that you can answer it.

There are typically three lectures per week. The definite timetable is available on Tabula[1], on the module website, or you can view your personal timetable on Tabula as well.

```
    class Applicative m => Monad m where
    return :: a -> m a
    return = pure

    (>>=) :: m a -> (a -> m b) -> m b
```

[1] https://tabula.warwick.ac.uk/profiles/department/cs/timetables?modules=cs141

6-10 January

Week 1 exercises

Relevant exercises for this week are *Getting started*, *Functions*, and *Basic types*.

6 January

Lecture 1: Introduction

What is functional programming and why should we learn it?

Overview of programming paradigms & models of computation, examples of applications of functional programming, module overview, and recommended texts.

7 January

Lecture 2: Definitions & functions

How do we write simple programs in Haskell?

Definitions, basic arithmetic expressions, string values, boolean values, functions, using built-in functions, and basic pattern matching.

8 January

Lecture 3: Basic types

How does the compiler prevent us from writing bad software?

Basic types, function types, parametric polymorphism, constraints, and pairs.

13-17 January

Week 2 exercises

Relevant exercises for this week are *Using the standard library*, *Lists*, *List comprehensions*, *Recursive functions*, *Higher-order functions*.

13 January

Lecture 4: Lists

How do we use lists in Haskell?

Constructing lists, pattern-matching on lists, and list comprehensions.

14 January

Lecture 5: Recursive functions

How do we express loops without mutable state?

Writing recursive functions for basic types (numbers, lists/strings), defining built-in functions ourselves.

15 January

Lecture 6: Higher-order functions

Can we write functions which abstract over common behaviours?

Higher-order functions such as `map`, `filter`, etc., recursion primitives such as `foldr` and `foldl`.

20-24 January

Week 3 exercises

Relevant exercises for this week are *Data types* and *Using other libraries*.

20 January

Lecture 7: Data types & type aliases

How can we define our own types in Haskell?

Type aliases, data types, data constructors, pattern matching on custom data constructors, recursion on values of custom types.

21 January

Lecture 8: *Fun with* functions

What are some more examples of functions?

Extended examples of using and defining functions.

22 January

Lecture 9: Coursework I briefing

What is the first coursework about?

Demonstration of what the completed coursework will do, explanation of the rules, introduction to the skeleton code.

27-31 January

Week 4 exercises

Relevant exercises for this week are *Type classes*.

27 January

Lecture 10: Type classes

How can we restrict polymorphism and overload functions?

Ad-hoc polymorphism via type classes, built-in type classes, and type class constraints.

28 January

Lecture 11: *Fun with* type classes

How are type classes used in Haskell?

Examples of type class and their instances.

29 January

Lecture 12: Testing

What tools are there for testing and how do we use them?

Unit testing, property-based testing, and code coverage in Haskell.

3-7 February

Week 5 exercises

Recommended exercises for this week are *Proofs*.

3 February

Lecture 13: Reasoning about programs

Can we use formal reasoning techniques to prove properties about our programs?

Equational reasoning, proofs by induction.

4 February

Lecture 14: Reasoning about programs (cont.)

Can we use formal reasoning techniques to prove properties about our programs?

Equational reasoning, proofs by induction.

5 February

Lecture 15: Constructive induction

Can we use formal reasoning techniques to calculate more efficient programs?

Using induction to calculate faster functions.

10-14 February

Week 6 exercises

Relevant exercises for this week are *Lazy evaluation*, *Foldables*, and *Functors*.

10 February

Lecture 16: Lazy evaluation

What order are programs evaluated in?

Strict and lazy evaluation, calling conventions, benefits and disadvantages, and examples of lazy evaluation.

11 February

Lecture 17: Foldables

Are there any useful design patterns in functional programming?

`Foldable` type class, its motivation, and examples.

12 February

Lecture 18: Functors & applicative functors

Are there any other useful design patterns in functional programming?

Functors, applicative functions, applications of applicative functors.

17-21 February

Week 7 exercises

Relevant exercises for this week are *Applicative functors*.

17 February

Lecture 19: Functors & applicative functors (cont.)

Are there any other useful design patterns in functional programming?

Functors, applicative functions, applications of applicative functors.

18 February

Lecture 20: *Fun with* applicative functors

What can we do with applicative functors?

Applicative functors in action.

19 February

Lecture 21: Coursework II briefing

What is the second coursework about?

Demonstration of what the completed coursework will do, semantics of the programming language, introduction to the skeleton code.

24-28 February

Week 8 exercises

Relevant exercises for this week are *Effectful programming*.

24 February

Lecture 22: Sequential composition

How do structure programs in which one part of a program relies on the result of another part?

Some functions for the sequential composition of `Maybe` values.

Lecture 23: Sequential composition (cont.)

25 February

Are there other examples of sequential composition?

Some functions for the sequential composition of `State` values and some laws for sequential composition

Lecture 24: *Fun with* sequential composition

26 February

How is sequential composition used in practice?

Sequential composition in action.

Week 9 exercises

2-6 March

Relevant exercises for this week are *Input & output* and *Kinds*.

Lecture 25: Input and output

2 March

Can we write impure programs in a pure programming language?

The `IO` monad.

Lecture 26: *Fun with* IO

3 March

What do Haskell programs that make use of IO look like?

The `IO` monad in action.

Lecture 27: Type promotion & GADTs

4 March

How can we encode more information in types?

Phantom types, GADTs, singleton types, pattern matching with GADTs.

Week 10 exercises

9-13 March

Relevant exercises for this week are *Type-level programming*.

Lecture 28: Type families

9 March

How can we perform computation at the type-level?

Closed and open type families.

Lecture 29: *Fun with* type-level programming

10 March

What are some examples of how type-level programming is used?

Type-level programming in action.

11 March	**Lecture 30: Conclusions** *What have we learnt about functional programming?* Summary of the module, information about the exam, and other general information.
13 March	**Deadline: Coursework II**
Term 3	**Revision lectures** Student-selected topics from the previous lectures.
Term 3	**Exam** 2 hours. Answer any four out of six questions.

3 Background

Your literature review goes here. Hughes (1989) discusses the advantages of functional programming by exploring *e.g.* lists, trees, and noughts and crosses in a functional programming language.

4 Progress

Summarise the progress you have made so far. You can cross-reference other sections (Section 3).

5 Project management

Include a timetable (in 2 week chunks) for the remainder of the academic year, up until the submission deadline.

References

Hughes, John. Why functional programming matters. *The computer journal*, 32(2):98–107, 1989.