

High Level Programming Introduction

Tom Clarke: Spring 2018

EE3 & EIE3

t.clarke@ic.ac.uk

615 EE

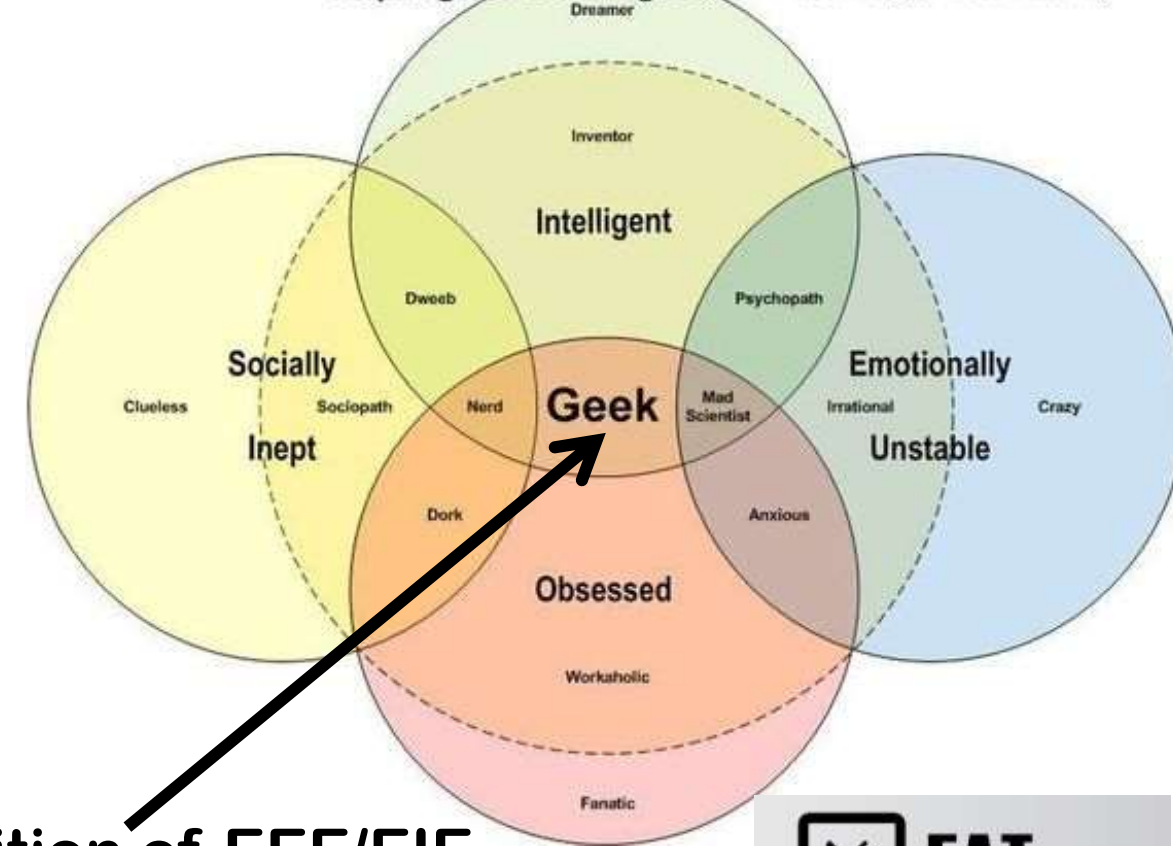
Prerequisites

- **Interest in programming**
 - ❖ You enjoy writing and analysing programs
- **Interest in concepts & maths** that underlies programming
 - ❖ You found recursion a really neat idea!
- If you have done significant **programming outside EEE/EIE classes** using Java/C#, etc:
 - ❖ You may have some object oriented (OO) "ways of thinking" to unlearn
 - ❖ This module will improve your ability program in all languages (including OO)
- If you have done **no programming outside EEE/EIE classes**
 - ❖ That is no problem for this module
 - ❖ EEE students should have been in the top 1/3 of the class for programming

What is a Geek?

<http://geek.travel/geek>

\ˈɡeɪk\
(noun)
An enthusiast or expert especially in a technological field or activity



Definition of EEE/EIE
surviving years 1 & 2??

NOT a good
idea!



Aims

➤ Learn to use functional programming language features

- ❖ Programs as functions
- ❖ Types as interfaces
- ❖ Rich (static) type systems

➤ Solve programming problems more efficiently

- ❖ Model problem domains with rich type system
- ❖ Use functional abstraction with higher-order functions
- ❖ Property-based testing

➤ Practice problem-solving in groups writing big programs

- ❖ Gain experience doing a significant a team programming project
- ❖ Define interfaces, modularise
- ❖ Top-down specification of code

➤ Have fun!

Coursework (weeks 1-4)

➤ Active Learning

- ❖ Learn through doing structured examples or project work
- ❖ Language reference is self-learnt
- ❖ Lectures provide introduction and motivation

➤ Programming worksheets to learn F# language

- ❖ Combine examples, analysis and coding
- ❖ Small tick-marked deliverables with tight deadline for each sheet provide structure

[From [Communications of ACM 2015](#)]

Be It Resolved: Teaching Statements Must Embrace Active Learning and Eschew Lecture

Last year, the *Proceedings of the National Academy of Science* published a meta-analysis of 225 studies. The conclusion appeared as the title of the paper, *Active learning increases student performance in science, engineering, and mathematics*.

Project work (weeks 6-11)

Timetable

➤ Teams of 3 or 4

➤ Last 6 weeks of Term

- ❖ 8 hours / week

➤ Projects

- ❖ Reimplement VisUAL – an ARM assembler and simulator
- ❖ Write a combinator graph-reduction based functional language

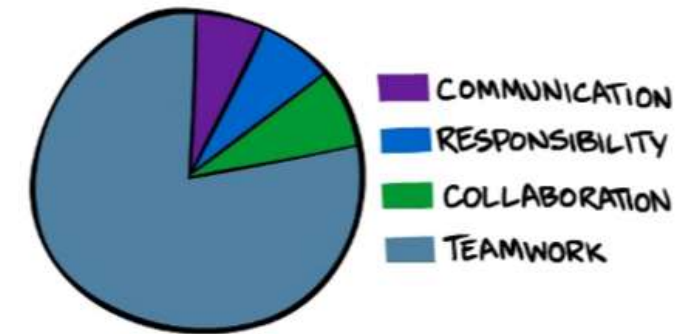
➤ Assessment

- ❖ Individual submitted code
- ❖ Group demo and presentation.

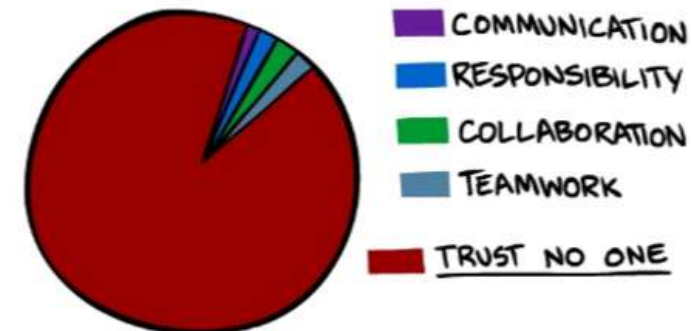
➤ Optional Technology used in project

- ❖ FABLE F# → Javascript transpiler
- ❖ Monaco editor programmer's editor component
- ❖ HTML for GUI (easy!)

WHAT GROUP PROJECTS ARE SUPPOSED TO TEACH YOU



WHAT GROUP PROJECTS TAUGHT ME



Content

- Functional Programming (FP) in F#
(50% of module)
- Lambda calculus what it is,
implementation
- Testing (property-based)
- OOP vs FP trade-offs: type systems and
coding style
- (Client-side web programming via F# -
> Javascript transpiler)

OOP vs FP

OOP makes code understandable
by **encapsulating** moving parts

FP makes code understandable
by **minimizing** moving parts

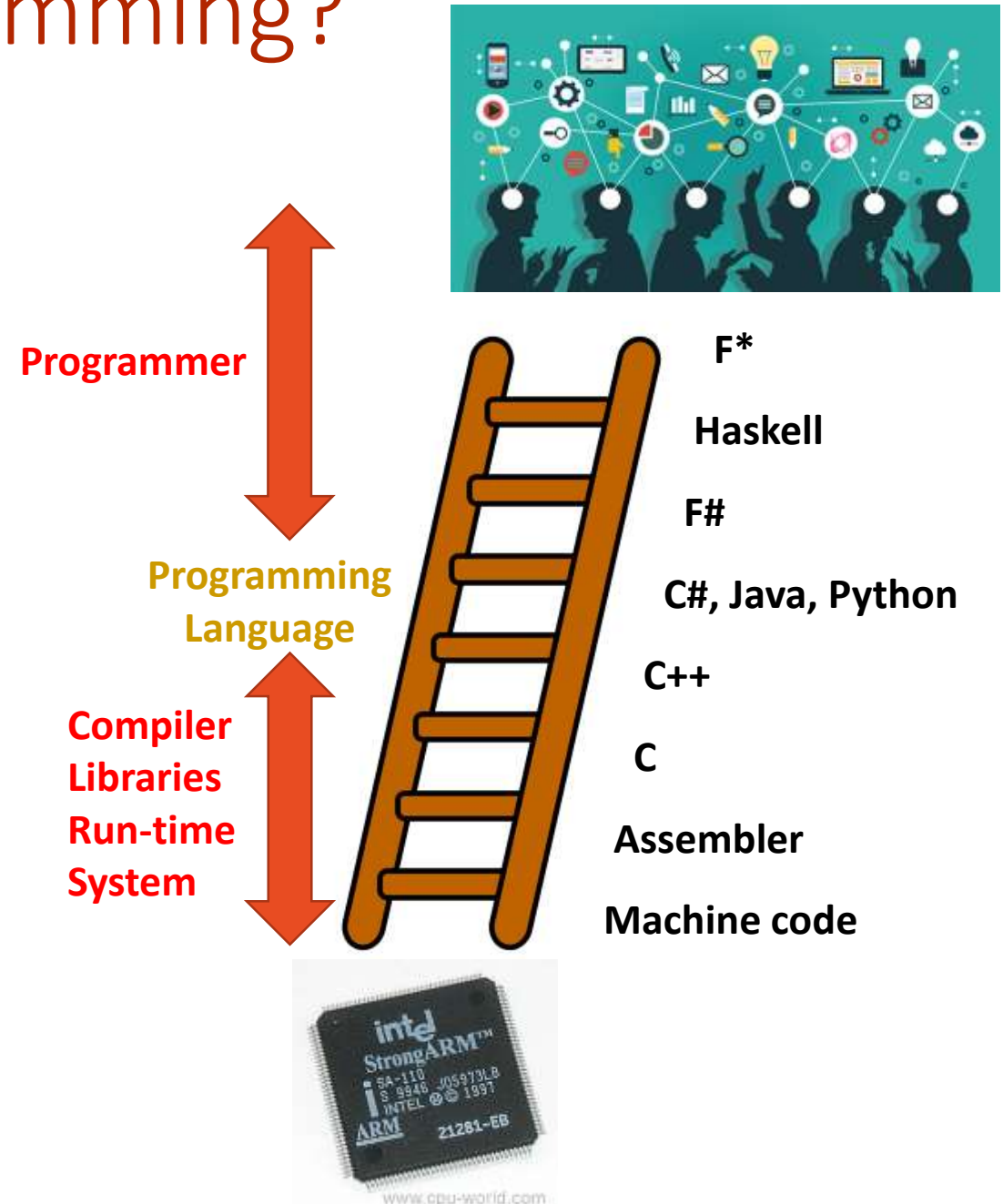
- Michael Feathers

[More](#)



What is High Level Programming?

- Let programming language do more work
- Support **abstraction** to make large/complex programs tractable
- Provide **type system** to document and specify operation and prevent errors
- Implement built-in **data structures** managed by compiler



Abstraction

Not
mutually
exclusive!

Three well-known paradigms:

- Procedural e.g. C
- OOP: Object Oriented Programming e.g. Java, C#
- FP: Functional Programming e.g. Haskell, FSharp

➤ Reusability

- ❖ Write once use many times

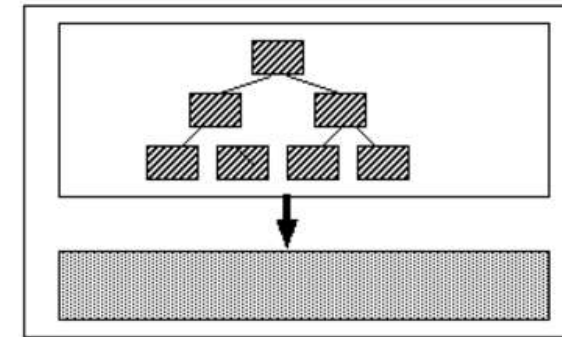
➤ Modularity

- ❖ Hide information - inside functions
- ❖ Hide information - inside modules
- ❖ Support Abstract Data Types (e.g. queues, lists)
- ❖ **OOP: encapsulate behaviour (state change) with state**

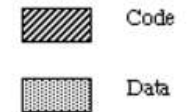
➤ Composability (make it easy to combine small parts)

- ❖ Reduce side effects
- ❖ Reduce global state
- ❖ **FP: Reduce local state**
- ❖ **FP: Allow functions as data**
- ❖ **OOP: Support making specific classes from general ones**

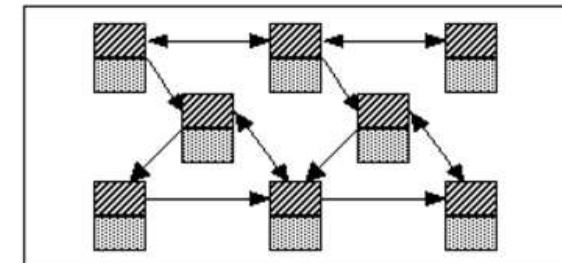
Procedural Languages



Computation involves code operating on Data



Object-Oriented Languages

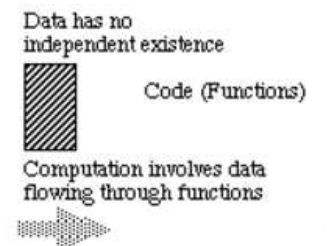
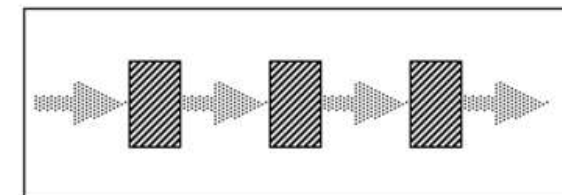


An object encapsulates both code and data



Computation involves objects interacting with each other

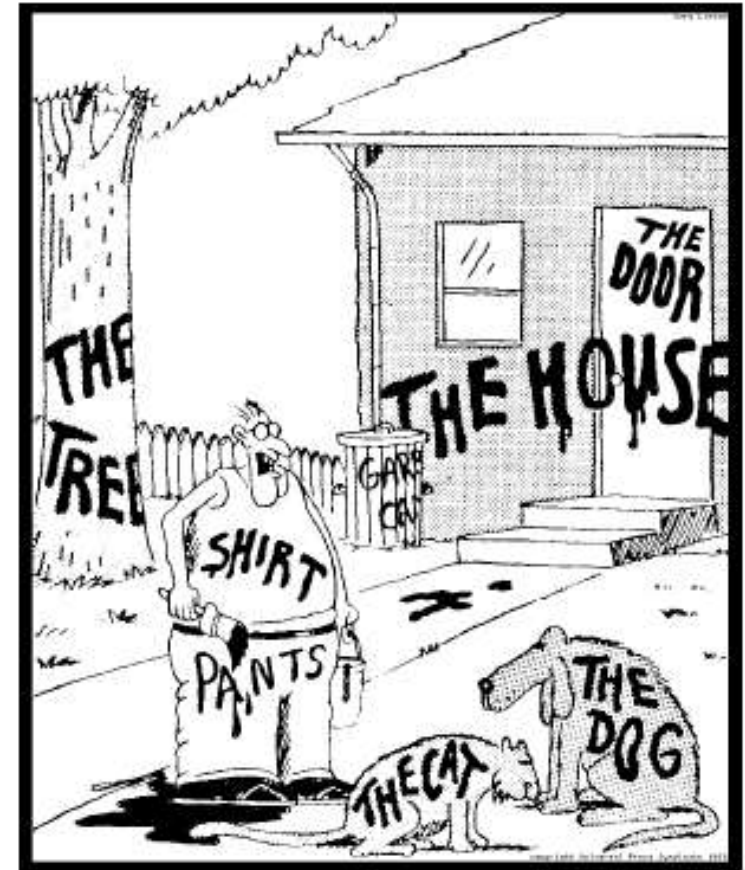
(Pure) Functional Languages



Type Systems

[45 min very accessible talk from developer: "What's wrong with Java's Type System"](#)

- Rich type system (Java, C#, F#)
 - ❖ Have lots of different types
 - ❖ Use types to differentiate and encapsulate data
 - ❖ Make it easy for types to model functions and data precisely
- Static vs dynamic type system
 - ❖ **Static**: (C, C++, Java, C#, F#) catch all type errors in compiler before running program
 - Types may restrict what you can do
 - ❖ **Dynamic** (Python, Matlab, Javascript): run-time catches type errors when you run program
 - Less protection against bugs, less documentation
 - More flexible (allows complex Javascript web frameworks)
 - Less type annotation "noise"
 - ❖ **Type inference: compiler works out most types from context!**
 - **Rescues static type system from type annotation "noise"**
 - **Flexibility remains an issue: trade type complexity for flexibility**

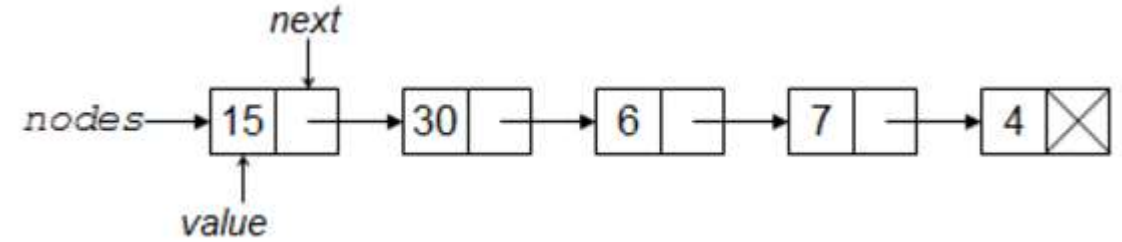


"Now! That should clear up a few things around here!"

**A rich, static, type system
(without type inference)**

credit Dr. Dobbs

Data Structures



➤ Good range of built-in data structures with convenient syntax

- ❖ List
- ❖ Array
- ❖ Map (lookup table)
- ❖ Set

Python does this well.
As do Java, C# (less well)

x = [15 ; 30 ; 6 ; 7 ; 4]

➤ High level memory management

- ❖ **No pointers needed**
- ❖ **Compiler allocates memory**
- ❖ **Run-time system frees memory (garbage collection)**
- ❖ **Memory allocation errors not possible!**

	KEYS	VALUES
	Jan	327.2
	Feb	368.2
	Mar	197.6
	Apr	178.4
	May	100.0
	Jun	69.9
	Jul	32.3
Aug	Aug	37.3
	Sep	19.0
	Oct	37.0
	Nov	73.2
	Dec	110.9
	Annual	1551.0

→ 37.3

Why learn F#?

- Big question - most of this module will be answering it!
- Short answer: functional languages give you a new perspective on programming **in any language**

- ❖ Improve programming style
- ❖ Learn new concepts

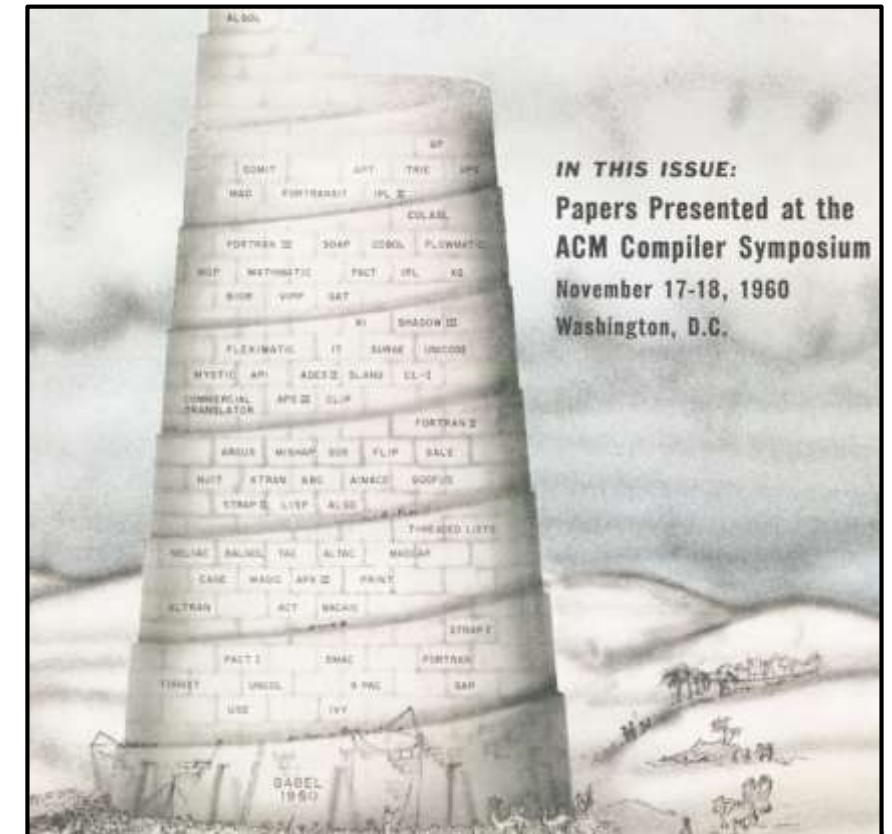
➤ What is F#?



- ❖ A real functional programming language used in industry with (highly paid) jobs
- ❖ Fully developed and with excellent tooling
- ❖ A practical language. Hybrid functional/OOP. Uses all of .NET libraries and ecosystem.

Almost as soon as people start talking about programming languages they also start criticizing the seemingly bewildering array and variety of such languages. The “Tower of Babel” metaphor was commonly invoked to describe this profusion of programming languages

The iconic reference to the Tower of Babel problem is the cover of the January 1961 issue of the Communications of the ACM, which featured the tower on the cover.



What is a functional language?

- Imperative programming
 - ❖ (1) Define and call functions
 - ❖ (2) Change the value of variables
- What happens if we get rid of (2)?
- Loops don't work!
 - ❖ get rid of them too!
 - ❖ use recursion instead of looping
- Programming looks like maths
 - ❖ What you want, not how you make it happen
- OK, so it's not that simple...

Imperative

```
int factorial(n) =  
{  
    int result = 1;  
    int i = 1;  
    while ( i < n) {  
        result = result * i;  
        i = i + 1;  
    }  
    return result;  
}
```

Functional

```
let factorial n =  
    let nums = listOfNumbersFrom1ToN  
    combineWithMultiply nums
```

Functional vs Imperative

➤ Imperative solution makes bugs easy!

- ❖ Order matters
- ❖ Loop limits easy to get wrong

➤ Functional solution requires new (old?) way of thinking about problem

➤ Functional solution requires complex data structures

➤ Functional is simpler

Imperative

```
int factorial(n) =  
{  
    int result = 1;  
    int i = 1;  
    while ( i < n) {  
        result = result * i  
        i = i + 1;  
    }  
    return result;  
}
```

Bug

Order
matters

Types are inferred by compiler
factorial: int -> int
nums: int list

Use let to emphasise
constant definition not
variable

Functional

```
let factorial (n) =  
    let nums = listOfNumbersFrom1ToN  
    combineWithMultiply (nums)
```

function is always one expression
"return" not needed

F# syntax: unfamiliar!

- Worksheet 1 will introduce new syntax as on right and much more
- Features:
 - ❖ `{ }` not used, blocks marked by indentation
 - ❖ `;` not needed
 - ❖ Function application no longer needs brackets
 - ❖ Shorthand for list of numbers
 - ❖ `x |> f`
 - `|>` is pipeline operator
 - feeds operand `x` to function `f`
 - same as `f x`
 - ❖ `List.reduce (*)`
 - *combineWithMultiply*

Functional

```
let factorial (n) =  
    let nums = listOfNumbersFrom1ToN  
    combineWithMultiply (nums)
```

F#

```
let factorial n =  
    let nums = [1..n]  
    nums |> List.reduce (*)
```

F# (shortened version)

```
let factorial n = [1..n] |> List.reduce (*)
```


Module goals: beyond learning a language

➤ Programming is much more than coding in a language:

- ❖ **Designing programs**
- ❖ **Testing programs**
- ❖ **Controlling dependency**
- ❖ **Modularising & specifying interfaces**

➤ What are the trade-offs in language choice?

- ❖ Static typing (contentious)
- ❖ Functional vs Object Oriented paradigms
- ❖ Tooling
- ❖ Libraries & re-usable code base
- ❖ Support for concurrency

theoretical issues
versus
pragmatic issues

Discover how the language
and type system affects code
readability and reusability

Learn how to write
concurrent programs
(the easy way)

Questions?