# Unit 1 - Introduction to Functional Programming

1. Stateful vs stateless computation
2. Lambda calculus semantics.
   1. Variables
   2. Lambda-abstractions/functions
   3. Function application
3. Introduction to F#, basic functionalities (main, print, …)
4. If-then-else, bindings, recursive bindings.
5. Examples and Exercises.

# Unit 2 – Types and Lists

1. Typed lambda calculus. Typing variables, lambda abstractions, function applications.
2. Lists and recursion.
3. Structural equality vs reference equality.
4. Exercises on lists.

# Unit 3 – Higher-order Functions

1. Curry, uncurry, pipe operator, function composition, map, fold, map2, fold2.
2. Case Study: SQL

# Unit 4 – Data Structures

1. Tuples and Unions in lambda-calculus (implementation shown in F#).
2. Tuples, Unions, and Records in F#.
3. Pattern matching,
4. Case Study: Expression evaluation

# Unit 5 – Advanced Data Structures

1. Recursive data structures (Trees, Lists with Cons and Empty, …)
2. Function records.
3. Case Study: Imperative Language with AST.

# Unit 6 – Drawing in Functional Programming

1. Function composition
2. Drawing lines
3. Drawing shapes
4. Using function composition for rendering.

# Unit 1 - Introduction to Functional Programming

1. Von Neumann model.
2. Computation as evaluating expressions
3. Semantics of untyped lambda-calculus
   1. Evaluation rule for variables
   2. Evaluation rule for functions
   3. Evaluation rule for function application
4. Introduction to F#
   1. Structure of a project: compilation file order, modules, import.
   2. Primitive types and values. Type inference.
   3. Example of program evaluation with respect to lambda calculus. Sum of two numbers.
   4. Shortcut for lambdas with multiple arguments. Comparison with lambda calculus.
5. Basic program constructs
   1. Let bindings. Evaluation of bindings in lambda calculus
   2. Recursive bindings and recursive functions. Recursion as a mean of looping in functional programming. **Examples:** factorial, Fibonacci sequence, integer division.
   3. If-then-else expression. Difference with imperative if-then-else. Just mention that it can be implemented in lambda calculus just with what we have. **Example:** Leap years.
6. **Exercises:**
   1. Return a string containing all numbers from 0 to n (n taken as input)
   2. Return a string containing all numbers from n to 0 (n taken as input)
   3. Return a string containing all numbers within a range (min and max taken as input)
   4. Return a string containing all even numbers within a range
   5. Draw a line of asterisks of a given length
   6. Draw a line made of a symbol taken as input of a given length
   7. Generate a binary string from an integer number
   8. Generate a string representing a number in an arbitrary base taken as input.

# Unit 2 – Types and Lists

1. Type rules of lambda calculus
   1. Typing variables
   2. Typing functions
   3. Typing function application
2. Structural Equality vs Reference Equality. Examples (C# lists vs F# lists)
3. Lists and recursive operations on lists (**do not use pattern matching. That comes later**)
   1. Length
   2. Sum
   3. Filter
   4. Even numbers from a list of integers
4. Tail recursion
5. **Exercises:**
   1. Return the last element of a list
   2. Reverse a list
   3. Find the nth elements of a list
   4. Take all the even elements of a list
   5. Check if a list is playndrome (true if equal to its reverse)
   6. Define a compress operation that removes all consecutive equal elements from a list. compress [a;a;a;a;b;b;c] = [a;b;c]