# 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. Exercises on lists.

# Unit 3 – Higher-order Functions and Data Structures

1. Pipe operator, function composition, map, fold, map2, fold2.
2. Tuples in lambda calculus and F#
3. Curry and Uncurry
4. Case Study: SQL

# Unit 4 – Data Structures

1. Discriminated unions in lambda calculus.
2. Discriminate unions and records in F#
3. Structural equality vs reference equality.
4. Pattern matching
5. Recursive data structures (Trees, Lists with Cons and Empty, …)
6. Case Study: Expression evaluation

# Unit 5 – Advanced Data Structures

1. Function records.
2. 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. Nested bindings. Shadowing.
   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 a positive number
   8. Generate a string representing a positive 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. Lists and recursive operations on lists (**do not use pattern matching. That comes later**)
   1. Length
   2. Sum
   3. Even numbers from a list of integers
   4. Mergesort
   5. Quicksort
3. Tail recursion (remake all the examples above with tail recursion)
4. **Exercises:**
   1. Return the last element of a list
   2. Reverse a list
   3. Append a list to another
   4. Find the nth elements of a list
   5. Check if a list is palindrome (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]
   7. Caesar’s cypher
   8. Vigenere’s cypher

# Unit 3 – Higher-order Functions

1. Pipe operator, function composition, map, fold, map2, fold2.
2. Tuples as higher-order functions in lambda calculus (tuple constructor, fst, snd)
3. Tuples in F#.
4. Case Study: SQL
5. **Exercises:**
   1. Use map to compute the cubic of each element of a list and collects the result in a list.
   2. Filter a list according to a given predicate.
   3. Given a list of functions and a list of elements, apply the function in position *I* to the element in the same position in the other list.
   4. Fold right
   5. Rewrite the sum of the elements of a list with only a fold.
   6. Multiply the elements of a list with only a fold.
   7. Length of a list as a fold.
   8. Append as a fold.
   9. Rewrite map as a fold.
   10. Rewrite filter as a fold.
   11. Reverse con fold right.

# Unit 4 – Data Structures

1. Match in F# as built-in abstraction.
2. Discriminate unions and non-recursive records in F#.
3. Structural equality vs reference equality.
4. Pattern matching
5. Recursive data structures (Trees, Lists with Cons and Empty, …)
6. Case Study: Expression evaluation
7. **Exercises:**
   1. Express the week days as a union and write a function
   2. Transform a list, possibly holding lists as elements into a `flat' list by replacing each list with its elements.
   3. Vectors with records (with operations).
   4. Shapes (Rectangle, Ellipse, Triangle, Polygon)
   5. Constructor for Circle and Square.
   6. Triangle area with vertices.
   7. Area for the shapes.

# Unit 5 – Advanced Data Structures

1. List with Cons, Empty. Map and fold with this definition.
2. Immutable Binary Search Tree. Find, Add, Remove.
3. String of a tree on console recursive.
4. Compare two trees.
5. Graph, DFS, BFS.
6. Type Classes with function records (Num<’a>, Show, IComparable<’a> as a record of functions).

# Unit 6 – Drawing in Functional Programming

1. Drawing lines
2. Drawing shapes
3. Fractals
4. Using function composition for rendering.
5. Game with update as a function record.