Introduction to OCaml

- Functional programming language with strong type inference
- Combines functional, imperative, and object-oriented styles
- Widely used in research, finance, and compiler development

Why Learn OCaml?

- Strong static typing improves reliability
- Concise syntax encourages clean code
- Good for symbolic computation, parsing, formal methods

Basic Syntax

- Variables are immutable by default
- Functions are first-class values
- Type inference reduces need for annotations

Data Types

- Primitive types: int, float, bool, char, string
- Tuples and Lists for collections

- Records and Variants for structured data

Pattern Matching

- Powerful way to deconstruct data
- Replaces complex if-else chains
- Common in handling lists, options, and variants

Functions in OCaml

- Anonymous functions: fun x -> x + 1
- Recursive functions: let rec fact n = if n=0 then 1 else n*fact(n-1)
- Higher-order functions: map, fold, filter

Modules

- Encapsulate related definitions
- Provide abstraction and namespace control
- Signatures define module interfaces

OCaml in Practice

- Used in compilers (Coq, ReasonML, etc.)
- Financial systems (Jane Street, Citadel)
- Static analysis and verification tools

In-Class Activity 1

- Write an OCaml function that computes the Fibonacci sequence.
- Bonus: Use pattern matching for the base cases.

In-Class Activity 2

- Define a variant type 'shape' with Circle, Square, and Rectangle.
- Write a function to compute the area using pattern matching.

Conclusion

- OCaml balances functional purity with practical features
- Encourages concise, reliable, and maintainable code
- A strong choice for systems requiring correctness and efficiency