Recursion:

* A function that calls itself until a base case is reached.
* Use **if** statement or pattern matching to define base case.
* Example: Factorial function using recursion

let rec factorial n =

if n <= 0 then 1

else n \* factorial (n-1)

Pattern Matching:

* Used to match a value against a set of patterns and execute corresponding code for each match.
* Syntax: **match value with pattern1 -> code1 | pattern2 -> code2 | ... | pattern\_n -> code\_n**
* Example: Function to extract the head and tail of a list using pattern matching

let head\_tail lst =

match lst with

| [] -> raise (Failure "empty list")

| hd :: tl -> (hd, tl)

Tail Recursion:

* A special case of recursion where the recursive call is the last expression in the function.
* Optimized by the compiler to use constant stack space.
* Example: Sum of a list using tail recursion

let rec sum lst accum =

match lst with

| [] -> accum

| hd :: tl -> sum tl (accum + hd)

Lists:

* A common data structure in OCaml represented using the syntax **[value1; value2; ...; value\_n]**.
* Lists can be manipulated using functions like **List.hd** (returns the head of the list), **List.tl** (returns the tail of the list), **List.map** (maps a function to each element of the list), etc.
* Example: Mapping a function to each element of a list

let square x = x \* x;;

let squares lst = List.map square lst;;

Example: List Processing with Pattern Matching

let rec sum l =

match l with

| [] -> 0

| x :: xs -> x + sum xs

Higher-Order Functions:

* Functions that take other functions as arguments or return functions as results.
* Useful for creating more abstract, reusable code.
* Example: Function to apply a given function **f** to each element of a list **lst**

let rec map f lst =

match lst with

| [] -> []

| hd :: tl -> (f hd) :: map f tl

Call Stacks/Trace:

* A call stack is a data structure that keeps track of function calls in a program.
* A trace is the sequence of function calls made in a program.
* The call stack allows you to track the flow of control in a program, including the arguments passed to functions and the values returned.

Fold\_right:

* A higher-order function that takes a binary function, a list, and an initial accumulator value.
* Applies the binary function to the elements of the list and the accumulator, in right-to-left order.
* Used to reduce a list to a single value by successively applying the binary function to each element of the list and the current accumulator value.