Control in Sequential Languages

- Structured Programming
  - Go to considered harmful
- Exceptions
  - "structured" jumps that may return a value
  - dynamic scoping of exception handler
- Continuations
  - Function representing the rest of the program
  - Generalized form of tail recursion
- Functions and evaluation order
  - Control evaluation order using function definitions and calls

Continuations

- Idea:
  - The *continuation* of an expression is "the remaining work to be done after evaluating the expression"
  - Continuation of expression `e` is a function; normally applied to `e`
- General programming technique
  - Capture the continuation at some point in a program
  - Use it later: "jump" or "exit" by function call
- Useful in
  - Compiler optimization: make control flow explicit
  - Operating system scheduling, multiprogramming
  - Web programming

Example: Roots of a quadratic equation

Version 0: without any continuations

```
exception notQuadratic;
exception imaginaryRoots;

fun equal(x:real, y:real) = x <= y andalso x >= y;

fun roots(a, b, c) =
  let
    val discBase = (b*b) - (4.0*a*c)
    val denom = 2.0*a
  in
    if equal(denom, 0.0) then raise notQuadratic
    else if discBase < 0.0 then raise imaginaryRoots
    else let
           val disc = Math.sqrt(discBase)
         in
           if disc > 0.0 then [(~b + disc)/denom, (~b - disc)/denom]
           else [~b/denom]
         end
  end;
```