Overview

- Calling Sequences
- Parameter Passing
- Exception Handling
- Coroutines
- Events
Coroutines

- Control abstraction that allows for
  - suspending execution
  - resuming where it was suspended

- For implementing non-preemptive multi-tasking
Coroutines (pseudo code)

us, cfs : coroutine

coroutine check-file-system()
    detach // create coroutine & return reference to caller

for all files:
    ...
    transfer (us)
    ...

main:
    us := new update-screen()
    cfs := new check-file-system()
    transfer (us)

coroutine update-screen()
    detach
    loop
        ...
        transfer (cfs)
Coroutines vs. Threads

- Explicit transfer of control (non-preemptive)
- Only one coroutine runs at a time

- Control flow transferred implicitly and preemptively
- Multiple threads may run concurrently
Coroutines vs. Continuations

- **Changes** every time it runs
- **Old program counter saved** when transferring to another coroutines
- **When transferring back**, continue where we left off

- **Once created, doesn’t change**
- **When invoking, old program counter is lost**
- **Multiple jumps to same continuation always start at some position**
Coroutines vs. Continuations

- Changes every time it runs
- Old program counter saved when transferring to another coroutines
- When transferring back, continue where we left off

- Once created, doesn’t change
- When invoking, old program counter is lost
- Multiple jumps to same continuation always start at some position

Both: Represented by a closure
(= code address + referencing environment)
Coroutines may call subroutines and create other coroutines

Each coroutine has its own function stack

- Second stack created when a routine creates a coroutine

Repeated creation of coroutines: “Cactus stack”
Example: Cactus Stack

Nesting of routine declarations:

\[ \text{static links} \]
Coroutines in Popular PLs

- **Natively** supported, e.g., in Ruby and Go
- Available as **libraries**, e.g., for Java, C#, JavaScript, Kotlin
- **Specialized variants**, e.g., in Python (generators)