Pipeline

Programming Style

Constraints

- We now have the power of ***functions*** (i.e., procedures with return values)

- No shared state between functions

- The larger problem is solved with functional composition (e.g., f∘g)

Notes

- Pure pipeline style uses **only** function composition
- No shared state means that our functions have **idempotence**, unlike the cookbook-style procedures
- Good for problems that allow for such constraints:
    - Quality issues such as testing and concurrency (idempotence means that testing results should be deterministic and tasks can be executed in isolation)

Notes

- Pure pipeline style also includes no function state from call to call
- The evolution in programming languages was from subroutines, to subroutines with inputs (procedures), to subroutines with inputs and outputs (functions)
- This style emerged in the 60s in the context of LISP
- Currently, Haskell is the language the embodies this style best