Introduction (1 of 2)

- Awk was designed by Alfred Aho, Peter Weinberger, and Brian Kernighan, at Bell Labs, in 1977.
- Awk is one of the early, and most popular, data-driven imperative PLs.
- A data-driven program has an implicit loop:
  - Read a line.
  - For each pattern: If the line matches the pattern, perform its action, perhaps producing output or transforming the line.
- An earlier data-driven PL was Sed (1974).
- Awk's string-processing was strongly influenced by Snobol (1962).
- Awk strongly influenced the very popular Perl (1987).
- Many PLs like Awk (e.g., Perl, Python, and Ruby) are called *scripting* PLs, which suggests *shell scripts*, but no one uses these PLs as a shell, like Bash.

Introduction (2 of 2)

- Awk is weakly and dynamically typed, but automatic conversions are often surprising.
- Awk is statically scoped, but function definitions cannot be nested, and there are no locals.
- Awk is (barely) higher-order. A string can be called as the name of a function.

Program structure (1 of 4)

- An Awk program is a set of function definitions, followed by a sequence of rules, which are pattern/action pairs.
- A function definition has a function name, formal-parameter names, and a body. There are no parameter types or return type.
- A call can have too few actual parameters. This can simulate local variables. A call with too many actual parameters produces a warning.
- A *pattern* is any expression, evaluated as a boolean, but is often just a regular expression to match against an input line.
- An *action* is a block of statements.

Program structure (2 of 4)

- After a pattern matches, and its action is performed, computation continues with the next rule.
- However, the `next` statement immediately goes back to the first rule and the next input line.
- An empty pattern always matches. An empty action prints the line.
- Patterns `BEGIN` and `END` are handy for initialization and finalization.

Program structure (3 of 4)

- Here's a simple example:

  `pub/sum/awk/sum2`
- This program has no function definitions and two rules.
- The first rule's pattern matches lines that contain exactly one integer. Its action sums them. The variable `$1` is the first field of the line. The variable `$0` is the whole line.
- The second rule prints the sum, after all lines have been processed.

Program structure (4 of 4)

- Here's another example:

  `pub/sum/awk/sum1`
- This program has one function definition and one rule.
- Since there are no rules with "regular" patterns, Awk's normal line-reading behavior is disabled.