Today’s lecture
• Continue looking at the details of Mathematica
• Closeup on various control structures from last lecture.

Review of Lecture 13
• Looked at Mathematica commands in more detail
  – Looked at variables and scope/assignments
  – IO commands: Import and Export
    • Open/Close
    • Read/Write
  – Character strings
  – Translated poly.f from FORTRAN to Mathematica
Control Statements: If

- If statement: Full generic form has three branches:
  \[ \text{If[logicalExpression, true, false, undefined]} \]
- Logical expressions are of form:
  - \( a \ == \ b \) — a equals b
  - \( a \ === \ b \) — a same as b (useful for strings)*
  - \( a \ != \ b \) — a not equal to b
  - \( a \ != \ b \) — a not the same as b
  - \( a \ \text{Greater than,} \ b \) — Greater than, \( a \ \text{Greater than or equal,} \ b \)
  - \( a \ \text{Less than,} \ b \) — Less than, \( a \ \text{Less than or equal.} \)
  - Combined expressions can be constructed with \( \text{not} \) (\!), \( \text{and} \) (\&\&) and \( \text{or} \) (\|\|) e.g.,
    \[ \text{If[a==b \&\& c==d, ...]} \] (precedence rules apply)

If statements 02

- One, two, or three executable statements can be included within the If e.g.
  \[ \text{If[a == b, c = d]; If[a == b, c = d; e = f];} \]
  \[ \text{If[a == b, c = d; e = f; g = h]} \] (note semicolons)
- If a symbol has the value True or False, then only the variable need appear *
  \[ \text{state = True;} \]
  \[ \text{If[state, Print["Program is finished"]]} \]
- Code within nested If statements should be indented with the indentation increasing with each layer of If statements.
- Mathematica offers alternatives to nested If statements:
Logical Functions in Addition to If

- `Which[test1, value1, test2, value2, ...]`
  - `Which` evaluates each test until one yields True. Then it evaluates the value for that test and exits the `Which`.
- Can have a default for `Which` by making the last test `True`
  - One way to write it for more readability is:
    - `Which[test1, value1, test2, value2, ...]`

- `Switch[expr, form1, value1, form2, value2, ...]`
  - compares expression to each form and evaluates the one corresponding to first True expression

Loop Statements Do, Table, For

- Looping constructions in Mathematica come in several forms. The two main forms are:
  - `Do[body, {j, start, end, inc}]`
  - `Table[body, {j, start, end, inc}]`
- The difference is that `Do` returns nothing (just executes the `body`), while `Table` executes and returns the `body` as a list.
- The lists appearing in the second argument for both `Do` and `Table` are called "iterators", and they appear in other Mathematica functions as well. `j` starts at value `start` and continues incrementing by `inc` until `j` is greater than `end`. If `inc = 1` then it does not need to appear. If `start = 1` then it does not need to appear. If the `body` doesn't depend on `j`, then `j` does not need to appear.
- If `end < start` and `inc` is positive, then the code inside the loop is never executed.
Looping Statements 02

- \( j \) can be real or integer, but for machine independent results, integer variables are recommended.
- The code inside Do or Table loops should be indented.
- Loops may be nested by listing multiple iterators. The first iterator is indexed first. The index variable in the loop may be modified, or its value used outside of the loop (since Mathematica sets up a local variable specifically for each loop).
  - If you want to have interaction of the loop with an outside value for the iterator, use the While function.
- \texttt{For[start, test, inc, body]} executes \texttt{start}, then does sequences of \texttt{test, inc, body} until \texttt{test} is not True. However, see Chapter 57 of the Beginner's Guide.

Other Commands

- \texttt{Get[context`]} — Instructs Mathematica to load a set of packages that contain rules and definitions (\textit{e.g.} \texttt{<<jleGroup`})
- See example packages. For a package, a .m file of executable code is created by Mathematica in parallel with a notebook file (.nb) if instructed to do so.
Pattern Matching

- One of the powerful features in Mathematica is its ability to select expressions, or parts of expressions, on the basis of pattern matching. Some examples from the *Mathematica Book*:
  - `f[n_]`  
    f with any argument, named n
  - `f[n_, m_]`  
    f with any arguments, named n and m
  - `x^n_`  
    x to a power named n
  - `x_^(n_)`  
    anything named x to a power named n
  - `a_ + b_`  
    a sum of two expressions named a & b
  - `{a1_ + a2_}`  
    a two-element list with members named a1 and a2
  - `f[n_, n_]`  
    a function with two identical arguments named n

Constants

- Mathematica has several built-in parameters. Useful ones are:
  - Pi, Degree, E, I
- You can define other constants with an equation *(e.g. pi2 = Pi^2 or twoPi = 2 * Pi)*.
- Such constants, like all symbols, are only available in the context in which they have been declared (thus the use of include statements)
Data

- Data would normally be stored in lists (either in a notebook or read from a file) and can be initialized at any time.
- An example of data initialization is:

  (* Day of year number at start of each month, valid in non-leap year *)
  daysInMonth = {0, 31, 59, 90, 120, 151, 
  . 181, 212, 243, 273, 304, 334}

Local Variables within Modules

- You should assume that variables local to a module will have no values each time the module is called. They will simply be symbols, with no value assigned to them.
- If you want variables to retain their values, then either use global variables (implying their values will be retained) or work within a user-defined context. Example

  BeginPackage["userDefinedContext"]
  exported symbols
  Begin[
    code (with private symbols)
  ]
  End[]
  EndPackage[]

- Each time the context is entered variables will retain the value that it had at the end of the last call.