- A fortran program consists of 3 main sections
  - Header section; this is where you declare your variable types
  - Executable section; this is where all the executable statements live. This is also the longest section of your program
  - End section; this section consists simply of the END reserved word.

- Reserved words - these are words that have specific meanings to the compiler. You can't use these words for variable names.
  - when I write programs, I will use all caps to indicate reserved words

- We will use the gfortran compiler. This is not as accurate as I would like, but our old compiler reached end of life and a new commercial compiler is very expensive
  - you can install this compiler on your own computer

- We will write programs using the Notepad++ editor.
  - This is also free and can be installed on your own computer
  - It will also use syntax highlighting so you can more easily identify reserved words.

- Fortran is not case sensitive (ie, A is the same as a)
  - This is because of backwards compatibility requirements
  - Keyboards in the 1960's didn't have lower case characters.

Variables

- Fortran supports 6 intrinsic (built in) types of variables and also allows you to create your own
  - INTEGER - used for variables with no decimal parts
  - LOGICAL - true/false (also called boolean)
  - IEEE single precision real } for variables that
  - IEEE double precision real } decimal parts
  - IEEE single precision complex } for complex numbers $(a+bi)$
  - CHARACTER - for variables consisting of things like names, addresses, etc.
- we will only use 2 of these to start with
  - INTEGER and double precision
  - Given that double precision is about 15-16 decimal places, this seems like overkill
  - Recall that for +, −, * and /, the result of something like

$$a+b = a+b \pm 1 \text{ digit in last place.}$$

  - This effect is cummulative

$$a+b+c = (a+b \pm 1\text{digit}) + c$$
$$= a+b+c \pm 2 \text{ digits in last place}$$

  - After 10 operations, the error could be as large as 10 digits in the last place (= 1 digit in next to last place)

1.2345678901234 5 6

↑ 10^9 ops ↑↑ 10 ops, 100 ops, 1000 ops

10^9 ops is not very many for a large program.

  - we use double precision to ensure that the 3 or 4 digits we actually have remains at 3-4 digits over trillions of ops.

- Rules for variables
  - must start with a letter
  - can contain letters, numbers and the underscore _
  - maximum length is 32 characters
  - you should pick variable names that are indicative of what the variable represents.
  - not case sensitive, so cat, CAT, cAt, etc. are all the same variable
  - we will write programs in such a way that all variables must be declared (ie, specifically given a type of INTEGER or double).
- Symbols
  - we use the usual symbols (mostly) for arithmetic operations

    + addition
    - subtraction
    * multiplication
    / division
    ** exponentiation (Keyboards in the 60's didn't have the ^ symbol).

- order of operations
  - These are necessary in order to translate a statement like

$$y = \frac{x^2}{1-x^2} \quad \text{to} \quad y = x^2/(1-x^2)$$

1) operations in ( ) done first, left to right
2) exponents (**) are next, left to right
3) *, / are next, left to right
4) +, - are last, left to right

- Examples
  - $y = \frac{x^2}{z^2} \rightarrow$ `y = X**2 /Z**2`
  - $y = \frac{x^2}{z^2+6} \rightarrow$ `y = X **2 /( Z**2 + 6)`
  - $w = y^2 - 3x + \frac{1}{k-5} \rightarrow$ `w = y**2 - 3**x + 1/(k-5)`
  - $w = V^{2x^2} \rightarrow$ `w = V**( 2* X**2)`
- You can use as many sets of ( ) as you want, but try not to use too many since this can make your formula harder to decipher
  - $y = \frac{x^2}{z^2}$ = `X**2 /Z**2`
    = `(X**2)/(Z**2)`
    = `((X**2)/(Z**2))` ← still OK, but hard to read.
- a complex formula can be broken down into easier to manage pieces

  $w = V^{2x^2} \rightarrow$ `y = 2*X**2`
  `w = V**y`

- one main goal is to write <u>portable</u> programs (ie, give accurate results no matter which compiler / computer is used).
  - The key to this is following the fortran standards
  - The standards are kind of like a legal document that describe how the language should behave
  - There is much that the standards do not address and not following them can cause unpredictable results.

- we will follow the book topics, but one main change is that we will use double precision instead of single precision.

- There are some key differences that need to be understood and followed in order to ensure that the code remains double precision

- Specifying constants in your program
  - when you need to use large or small numbers, you can use scientific notation

1.2345×10^{15} → 1.2345e15

6.432×10^{-8} → 6.432e-8

  - a = 1 ← no decimal point, so this is an INTEGER 1

  - b = 1.3 ← has a decimal point, so this is a single precision 1.3 (this is the dangerous case).

  - c = 1.3d0 ← This is a double precision 1.3
    Think of this as

$1.3 \times \underbrace{10^{0}}_{\text{double precision } 10^{0}}$

  - b and c above are not the same although the differences are subtle. Computations performed using b and c will behave differently and give different results.

- For the previous example

1.2345×10^{15} → 1.2345e15 (single, not what we want to do)

→ **1.2345d15** (correct double precision)

6.432×10^{-8} → 6.432e-8 (single, not correct)

→ **6.432d-8** (double, correct)

- IF the constant has digits after the decimal point, you need to use the 'd' descriptor to properly assign a double precision value

except for powers (sometimes)

Examples

incorrect	correct
a = 6.92	a = 6.92d0
b = 0.0032	b = 0.0032d0 or 3.2d-3