Cheat Sheet - Strings

This sheet is intended to provide an overview of Fortran strings and their use

- 00 Terminology
  - Character can mean:
    - the *TYPE* used for strings/letters in Fortran
    - a single element of a string, a letter
  - String - a sequence of characters
- 01 Fixed Length Strings
  - CHARACTER(LEN=n) :: name
    - A string of length n where n is compile time fixed
    - Can hold up to n letters (inclusive, no need for extra space for a terminator)
    - Spare space the the end is full of blanks
  - CHARACTER(LEN=n) :: name = "Hello"
    - Creates a string and sets it to the literal value "Hello" (provided n>= 5)
  - PRINT*, name
    - Output to screen, *inclusive* of blanks
  - LEN(name) -> 7
    - LEN function returns the length of the string (including any blanks etc)
  - Fortran has no concept of single characters, only length-1 strings
  - CHARACTER, DIMENSION(5) :: not_a_string
    - This defines an array of characters, i.e. length-1 strings. Setting this to "Hello" does not work as above but rather gives "HHHHH". "Hello" is turned into a length-1 string and each element of the array set to this string. Do not confuse this with "true strings"!
  - If you do want an array of strings, you can do

- CHARACTER(LEN=10), DIMENSION(:), ALLOCATABLE :: string_array
  - Allocatable array of length-10 strings
  - Access by array index first, then by string index e.g.
    - string_array(1)(1:10) = "A string !"
    - See next segment for string access details

- 02 Accessing Strings
  - Looks a lot like arrays
  - CHARACTER(LEN=7) :: name = "Hello"
    - name(1:1) -> "H" ! A Length 1 string
    - name(2:3) -> "el" ! A Length 2 string
    - name(5:7) -> "o " ! A Length 3 string with 2 trailing blanks
    - name(1:) -> "Hello " ! A Length 7 string
    - name(:5) -> "Hello" ! A Length 5 string
  - Characters can be set by assigning to these substrings
    - name(2:3) = "aa" -> "Haalo " !Stays Length 7
  - String size is fixed and can't change
    - name = "12345678" -> "1234567" ! Stays Length 7
      - Trying to store over-long data is not an error but leads to SILENT DATA LOSS if you actually needed all of those characters!
    - name = "12345" -> "12345 " ! Assignment always adds those blanks!
    - Might be tempted to use a range - but this introduces potential for miscounting errors
      - name(1:8) = "12345678" -> *COMPILE ERROR - Substring out of range*
      - name(1:8) = "1234567" -> Also compile error even though data is valid

- 03 Printing Strings
  - PRINT*, name

- Prints string including all trailing blanks

- PRINT '(A)', name
  - Prints string with explicit format code, still includes trailing blanks
- PRINT '(A5)', name
  - Prints 5 characters of given string starting from the left
- See below for how to avoid seeing blanks without having to count characters all the time!

- 04 Manipulating Strings
  - CHARACTER(LEN=7) :: name = " Hello" -> " Hello " !(Note spaces)
  - ADJUSTL(name) -> "Hello " ! Left-justifies i.e. moves all characters to the left. Length stays 7!
  - ADJUSTR(name) -> " Hello" ! Right justifies i.e. moves all characters to the right. Length stays 7!
  - TRIM(name) -> " Hello" ! Removes trailing blanks. Length now 5!
  - TRIM(ADJUSTL(name)) -> "Hello" ! Remove both leading and trailing blanks
  - "Hello "//"World" -> "Hello World" ! String concatenation operator
- 05 Passing Strings to Functions
  - String dummy variables (parameters to functions) can have unspecified length and will be just "as long as the passed string is". This uses the '*' for the length

    SUBROUTINE mysub(name)

     CHARACTER(LEN=*) :: name

     PRINT*, LEN(name)

    END SUBROUTINE

    mysub("Hello") -> 5 ; mysub("123456789") -> 9 ;etc
  - This is best just used as magic because the explanation for why '*' and why this isn't like arrays is messy. Recall that strings are not arrays of characters (no DIMENSION attribute)
- 06 Filling Strings

- WRITE(name, '(F8.3)'), 1234.5678 -> 1234.568
  - Write the given number into the string 'name' using the given format
  - Rounding can occur (and does in this example)
  - String must be large enough to hold the result or a runtime error occurs
  - May also encounter WRITE with a '*' or a number as the first argument - these write to the default output (the screen) or to a logical unit number respectively
- Creating a formatted message string (assuming name is long enough to hold the entire result):
  - WRITE(name, '(A,F8.3,A)') "The value is ", 1234.5666, "m"
  - WRITE(name, '("The value is ", F8.3, " m")' 1234.5666 ! Note use the other kind of quotes around your literal strings
- Get user input as a string (string must be large enough to hold input
  - READ(*, fmt) name
    - Read from the command line. Format can be '*' to use the default or 'A{n}' to limit the characters read
- Convert string data into a given variable type
  - READ(name, fmt) data
  - Reads from the string "name", into the variable "data"

- 07 Allocatable Strings
  - Fortran 2003 introduced run-time sizeable, i.e. allocatable, strings. Because these are strings, NOT arrays of characters, their syntax is a bit different to arrays
  - CHARACTER(LEN=:), ALLOCATABLE :: name
    - An allocatable string. Length is "deferred" - to be given later
  - ALLOCATE(CHARACTER(LEN=10):: name)
    - Allocator has to refer to the length parameter, not create an array
    - This is basically specifying which allocator function to use and its parameter
    - If you know OO Fortran this makes more sense

- ALLOCATE(CHARACTER(10) ::name)
  - As long as allocator is supplied, can omit the LEN specifier but this might be harder to read
- Automatic allocation, re-allocation to a new size etc can occur as for arrays. Unfortunately this can't happen in READ statements, which limits is useful-ness

- 07 Other Useful Functions
  - INDEX(str, substr)
    - Returns the index of the first occurence of substring in given string
    - Optional final logical parameter *back* to search from end to start
  - REPEAT(substr, n)
    - Returns a string consisting of n repeats of substr
  - SCAN(str, chars)
    - Returns the index of the first occurrence of any of the chars given (chars are given as a string)
    - e.g. SCAN("abcd", "cd") -> 3 ; SCAN("abcd", "efgh") -> 0
    - Again optional final *back* param to search from end to start
  - VERIFY(str, chars)
    - Verify that only the characters in chars are present in the string. Otherwise returns the index of the first character in str that is not in the given list (chars are again given as a string)
    - Again optional final *back* param to search from end to start (i.e. give the last occurrence of a character not in chars