Functions

In algebra, when prescribing a function f of the form

$$y = f(x)$$

we must specify the type of the argument(s) (the input to the function – in this case the single variable x) and the type of the return value y. We must also specify how the function acts upon its argument(s).
For example, the Euclidean norm in 3-D:

$$f : \mathbb{R}^3 \rightarrow \mathbb{R}$$

$$f : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \sqrt{x^2 + y^2 + z^2}$$

Functions in Fortran

The are several allow forms for the syntax of a function declaration in Fortran. We shall use:

```
rtype   FUNCTION   fname(args)   RESULT(rvar)
    function-body
END  FUNCTION   fname
```

where:

- `fname` is the function name
- `args` are the arguments supplied to the function
- the return value is of type `rtype` and is stored in `rvar`

If the function is permitted to call itself (i.e. if the function is *recursive* then the `FUNCTION` keyword in the top line must be preceded by the keyword `RECURSIVE` .

Fortran function code

```
!<type> FUNCTION fname(vars) RESULT(var)
REAL(8) FUNCTION mynorm(xv, yv, zv) RESULT(res)
  REAL(8), INTENT(IN) :: xv, yv, zv ! INTENT attribute for c
  REAL(8)               :: a          ! a variable internal to
  a = xv**2 + yv**2 + zv**2           ! do some arithmetic
  res = SQRT(a)                       ! assign the dummy resul
END FUNCTION mynorm

PROGRAM norm1
  IMPLICIT NONE
  REAL(8) :: a, x, y, z               ! coordinates
  REAL(8), EXTERNAL :: mynorm         ! function name
  x =  3.1
  y =  4.2
  z = -5.3
  a = mynorm(x, y, z)                 ! call function
  PRINT*, a
END PROGRAM norm1
```

Recursive function example

```
! <type>  RECURSIVE FUNCTION fname(vars) RESULT(var)
INTEGER(8) RECURSIVE FUNCTION myfac(n) RESULT(res)
  INTEGER(8), INTENT(IN) :: n ! INTENT attribute for dummy v
  IF (n < 2) THEN
    res = 1
  ELSE
    res = n * myfac(n - 1)
  END IF
END FUNCTION myfac

PROGRAM factorial
  IMPLICIT NONE
  INTEGER(8) :: m, n
  INTEGER(8), EXTERNAL :: myfac
  READ*, n
  m = myfac(n)                          ! call function
  PRINT*, n, ' factorial is ', m
END PROGRAM factorial
```

Subroutines

Subroutines are like functions but without a return value. The syntax is:

```
SUBROUTINE    sname(args)
    subroutine-body
END  SUBROUTINE    sname
```

where

- `sname` is the subroutine name
- `args` are the arguments supplied to the subroutine

If the subroutine is recursive then the in the top line the keyword `SUBROUTINE` must be preceded by the keyword `RECURSIVE` .

Fortran subroutine code

```
SUBROUTINE pairsort(x, y)
  INTEGER, INTENT(INOUT) :: x, y ! x, y can be read and modi
  INTEGER :: n                   ! local variable
  IF (x > y) THEN                ! if x > y then swap them
    n = x
    x = y
    y = n
  END IF
END SUBROUTINE pairsort

PROGRAM sort0
  IMPLICIT NONE
  INTEGER :: a, b                ! two integers
  READ*, a, b                    ! read them from users
  PRINT*, 'a = ', a, ' b = ', b ! print a, b to screen
  CALL pairsort(a, b)            ! call subroutine
  PRINT*, 'a = ', a, ' b = ', b ! print a, b again
END PROGRAM sort0
```

Dummy variables

- If a subroutine or function modifies one its arguments, then the calling variable corresponding to that argument itself is modified (example in previous slide).
- The argument (and return value) variables in subroutines/functions are therefore referred to as *dummy variables* because they're markers for the calling variables.
- Whether you're obliged/permitted to modify the arguments of a subroutine/function is controlled by the `INTENT` keyword:

```
SUBROUTINE my_routine(x, y, z)
  INTEGER, INTENT(IN)    :: x ! MUST NOT be assigned
  INTEGER, INTENT(OUT)   :: y ! MUST be assigned
  INTEGER, INTENT(INOUT) :: z ! MAY be assigned
```

Function or subroutine?

- Since it's possible to
  - ignore the return value of a function
  - modify the arguments supplied to a function or subroutine

  anything you can do with a subroutine you can do with a function and vice-versa.
- My recommendation:
  - if you want to evaluate what we'd refer to mathematically as a function, i.e. something of the form $x = f(\{a_i\})$, where the arguments $\{a_i\}$ are unmodified as a result of evaluating f, then use a Fortran FUNCTION, declaring the arguments as input only
  - otherwise use a Fortran subroutine