Subprograms

More Inconsistent Terminology

- Function
  - Ideally, a function takes any number of *parameters* (up to a system-dependent limit) and returns a single item.
  - In practice, this is too limiting so programmers and languages have ways around it.
- Subroutine
  - A subroutine takes any number (up to some limit) of parameters and returns any number (up to some limit) of values.
  - The terminology is used formally only in Fortran (as far as I know) but informally in other languages.

More Terminology

- Procedure, Subprogram
  - Sometimes used as generic terms for function or subroutine
- Method
  - A procedure that is only accessible through a defined type (a class or object).

Fortran Functions

Syntax

```
FUNCTION myfunc(param1,param2,param3,param4)
  <type>                    :: myfunc
  <type>, INTENT(in) :: param1, param2, param3
  <type>, INTENT(in) :: param4
        statements
        myfunc=whatever
       return          ! Optional unless a premature return
END FUNCTION myfunc
```

Renaming Result

- Use the result clause

  ```
  function sum(top) result(s)
  ```

- Mostly used for recursive functions

Fortran Subroutines

- Syntax

```
SUBROUTINE mysub(param1,param2,param3)
<type> INTENT(in)      :: param1
<type> INTENT(out)     :: param2
<type> INTENT(inout) :: param3
      statements
      return          ! Optional unless premature
END SUBROUTINE mysub
```

Invoking Functions/Subroutines

- Function
  - Invoke by its name

```
x=myfunc(z,w)
y=c*afunc(z,w)
```

  A function is just like a variable except it cannot be an *lvalue* (appear on the left-hand side of =)
- Subroutine
  - Use the call keyword

```
CALL mysub(x,y,z)
```

More Fortran: Interfaces

- If your subprogram is not in a module (more later) you SHOULD provide an INTERFACE
- Equivalent to a function prototype in other languages
- If there is an explicit (you write) or implicit (in a module) interface, the compiler will check that the *number* and *type* of the arguments agree in the subprogram and in the call.

Interfaces

- Syntax

```
INTERFACE
    function myfunc(x,y,z)
      implicit none
      real :: myfunc
      real :: x,y
      complex :: z
    end function myfunc
END INTERFACE
```

Interfaces (Continued)

```
INTERFACE
    SUBROUTINE mysub(x,y,z)
        use mymod
        implicit none
        <type> :: x
        <type> :: y,z
    END SUBROUTINE mysub
END INTERFACE
```

Interfaces (continued)

- Only one interface block is required per program unit. It is nonexecutable and goes with the declarations.

```
INTERFACE
    function mysub
        Blah blah
    end function mysub
    subroutine mysub1
    end subroutine mysub1
    subroutine mysub2
    end subroutine mysub2
END INTERFACE
```

Optional Arguments

- Syntax:

```
subroutine mysub(x,y,z)
implicit none
intent (in)                 :: x
intent(in), optional :: y,z
```

- If not present in the argument list, they don't exist. Use the present intrinsic to check.

Anonymous Functions

- Fortran: inline functions
  - Marked for deletion but you may see them in old code
  - Uses are rather limited but occasionally they are handy
  - Must be expressible as a single line
  - Not really anonymous (they need a name) but behave like one
  - Are nonexecutable and go with the declarations. Exist only in the program unit in which they are declared.

Variable Scope

- The *scope* of a variable is the range over which it has a defined value. In Fortran, scope is defined by the program unit. In Python, the scope of a variable is the code block within which it is first referenced. So a calling program may have a variable named x, and a function may also have a variable named x, and if x is not an argument to the function then it will be distinct from the x in the main program.

Fortran: Scope Example

```
x=20.
call sub(x)
etc.
subroutine sub(y)
real, intent(inout) :: y
real                :: x
   x=10.
   y=30.
end subroutine sub
```

Fortran: CONTAINS

- The contains keyword extends the scope into the contained program unit.
- The end of the “container” must *follow* the end of the “containee”
- A contained subprogram can access all the variables in the container except those that are explicitly passed.
- Interface is implicit, should not be explicit

CONTAINS example

```
program myprog
implicit none
real  :: x,y,z
      x=5.; y=10.
      call mysub(z)
      contains
      subroutine mysub(w)
            real, intent(inout) :: w
            w=x+y
      end subroutine mysub
end program myprog
```

Recursion

- When a function calls itself this is called *recursion*.
- Make sure you have a stopping condition that *will be* met!!
- Fortran requires the `recursion` keyword.
- Don't try this until you are completely comfortable with regular functions!! But it's sometimes the best way to express an algorithm.

Everybody's Favorite Recursion

- Fibonacci numbers:
- F(0)=0
- F(1)=1
- F(2)=F(1)
- F(3)=F(2)+F(1)
- F(4)=F(3)+F(2)
- …
- F(n)=F(n-1)+F(n-2)

Fortran

```
recursive function fib(n) result(fb)
implicit none
integer, intent(in) :: n
integer                    :: fb
      if ( n<0 ) then
            print *,'Illegal value'; fb=-1
            return
      else if (n==0 ) then
            fb=0
      else if (n==1) then
            fb=1
      else
            fb=fib(n-1)+fib(n-2)
      endif
end function fib
```