FORTRAN - MODULES

A module is like a package where you can keep your functions and subroutines, in case you are writing a very big program, or your functions or subroutines can be used in more than one program.

Modules provide you a way of splitting your programs between multiple files.

Modules are used for:

- Packaging subprograms, data and interface blocks.
- Defining global data that can be used by more than one routine.
- Declaring variables that can be made available within any routines you choose.
- Importing a module entirely, for use, into another program or subroutine.

Syntax of a Module

A module consists of two parts:

- a specification part for statements declaration
- a contains part for subroutine and function definitions

The general form of a module is:

```
module name
   [statement declarations]
   [contains [subroutine and function definitions] ]
end module [name]
```

Using a Module into your Program

You can incorporate a module in a program or subroutine by the use statement:

```
use name
```

Please note that

- You can add as many modules as needed, each will be in separate files and compiled separately.
- A module can be used in various different programs.
- A module can be used many times in the same program.
- The variables declared in a module specification part, are global to the module.
- The variables declared in a module become global variables in any program or routine where the module is used.
- The use statement can appear in the main program, or any other subroutine or module which uses the routines or variables declared in a particular module.

Example

The following example demonstrates the concept:

```
module constants
implicit none

   real, parameter :: pi = 3.1415926536
```

```
   real, parameter :: e = 2.7182818285

contains
   subroutine show_consts()
      print*, "Pi = ", pi
      print*,  "e = ", e
   end subroutine show_consts

end module constants


program module_example
use constants
implicit none

   real :: x, ePowerx, area, radius
   x = 2.0
   radius = 7.0
   ePowerx = e ** x
   area = pi * radius**2

   call show_consts()

   print*, "e raised to the power of 2.0 = ", ePowerx
   print*, "Area of a circle with radius 7.0 = ", area

end program module_example
```

When you compile and execute the above program, it produces the following result:

```
Pi = 3.14159274
e =  2.71828175
e raised to the power of 2.0 = 7.38905573
Area of a circle with radius 7.0 = 153.938049
```

Accessibility of Variables and Subroutines in a Module

By default, all the variables and subroutines in a module is made available to the program that is using the module code, by the **use** statement.

However, you can control the accessibility of module code using the **private** and **public** attributes. When you declare some variable or subroutine as private, it is not available outside the module.

Example

The following example illustrates the concept:

In the previous example, we had two module variables, **e** and **pi.** Let us make them private and observe the output:

```
module constants
implicit none

   real, parameter,private :: pi = 3.1415926536
   real, parameter, private :: e = 2.7182818285

contains
   subroutine show_consts()
      print*, "Pi = ", pi
      print*, "e = ", e
   end subroutine show_consts

end module constants


program module_example
use constants
implicit none
```

```
   real :: x, ePowerx, area, radius
   x = 2.0
   radius = 7.0
   ePowerx = e ** x
   area = pi * radius**2

   call show_consts()

   print*, "e raised to the power of 2.0 = ", ePowerx
   print*, "Area of a circle with radius 7.0 = ", area

end program module_example
```

When you compile and execute the above program, it gives the following error message:

```
   ePowerx = e ** x
   1
Error: Symbol 'e' at (1) has no IMPLICIT type
main.f95:19.13:

   area = pi * radius**2
   1
Error: Symbol 'pi' at (1) has no IMPLICIT type
```

Since **e** and **pi,** both are declared private, the program module_example cannot access these variables anymore.

However, other module subroutines can access them:

```
module constants
implicit none

   real, parameter,private :: pi = 3.1415926536
   real, parameter, private :: e = 2.7182818285

contains
   subroutine show_consts()
      print*, "Pi = ", pi
      print*, "e = ", e
   end subroutine show_consts

   function ePowerx(x)result(ePx)
   implicit none
      real::x
      real::ePx
      ePx = e ** x
   end function ePowerx

   function areaCircle(r)result(a)
   implicit none
      real::r
      real::a
      a = pi * r**2
   end function areaCircle

end module constants


program module_example
use constants
implicit none

   call show_consts()

   Print*, "e raised to the power of 2.0 = ", ePowerx(2.0)
   print*, "Area of a circle with radius 7.0 = ", areaCircle(7.0)
```

```
end program module_example
```

When you compile and execute the above program, it produces the following result:

```
Pi = 3.14159274
e = 2.71828175
e raised to the power of 2.0 = 7.38905573
Area of a circle with radius 7.0 = 153.938049
```