Intrinsic Functions Outline

See *Programming in Fortran 90/95*, 1st or 2nd edition, Chapter 10 section 10.4, and especially Chapter 17.

Functions in Mathematics

"In mathematics a **function** is a rule that assigns to each element of a set an element of the same or of another set." (Bers & Karal, *Calculus*, 2nd ed, Holt, Reinhart & Winston, 1976, p. 50.)

So, for example, if we have a function

$$f(x) \; = \; x \; + \; 1$$

then we know that

$$\begin{array}{lcrclcr}
 & & & \ldots & & & \\
f(-2.5) & = & -2.5 & + & 1 & = & -1.5 \\
f(-2) & = & -2 & + & 1 & = & -1 \\
f(-1) & = & -1 & + & 1 & = & 0 \\
f(0) & = & 0 & + & 1 & = & +1 \\
f(+1) & = & +1 & + & 1 & = & +2 \\
f(+2) & = & +2 & + & 1 & = & +3 \\
f(+2.5) & = & +2.5 & + & 1 & = & +3.5 \\
 & & & \ldots & & &
\end{array}$$

Likewise, if we have a function

$$a(y) \; = \; |\, y \,|$$

then we know that

$$\begin{array}{lcl r lcl}
 & & & \ldots & & & \\
a(-2.5) & = & | & -2.5 & | & = & 2.5 \\
a(-2) & = & | & -2 & | & = & 2 \\
a(-1) & = & | & -1 & | & = & 1 \\
a(0) & = & | & 0 & | & = & 0 \\
a(+1) & = & | & +1 & | & = & 1 \\
a(+2) & = & | & +2 & | & = & 2 \\
a(+2.5) & = & | & +2.5 & | & = & 2.5 \\
 & & & \ldots & & &
\end{array}$$

We refer to the thing inside the parentheses – in the first example it'd be *x*, and in the second example *y* – as the *argument* (or sometimes the *parameter*) of the function.

Functions in Fortran 90

In `my_number.f90`, we saw this:

```
  ...
  ELSE IF (ABS(users_number - computers_number) <= &
 &         close_distance) THEN
      PRINT *, "Close, but no cigar."
  ...
```

So, what does `ABS` do?

`ABS` is a function that calculates the *absolute value* of its argument. It is the Fortran 90 analogue of the mathematical function

$$a(y) = |y|$$

(the absolute value function) that we just looked at. So:

```
          ...
ABS(-2.5) → 2.5
ABS(-2)   → 2
ABS(-1)   → 1
ABS(0)    → 0
ABS(1)    → 1
ABS(2)    → 2
ABS(2.5)  → 2.5
          ...
```

Note: in this example, → denotes "evaluates to," or, in computing jargon, "*returns*." We say "`ABS` of -2 evaluates to 2" or "`ABS` of -2 returns 2."

An important distinction between a mathematical function and a Fortran 90 function: a mathematical function is simply a **definition**, while a Fortran 90 function **does stuff**. More on this presently.

A Quick Look at ABS

```
% cat abstest.f90
PROGRAM abstest
    IMPLICIT NONE
    PRINT *, "ABS(-2.5) = ", ABS(-2.5)
    PRINT *, "ABS(-2)   = ", ABS(-2)
    PRINT *, "ABS(-1)   = ", ABS(-1)
    PRINT *, "ABS( 0)   = ", ABS( 0)
    PRINT *, "ABS( 1)   = ", ABS( 1)
    PRINT *, "ABS( 2)   = ", ABS( 2)
    PRINT *, "ABS( 2.5) = ", ABS( 2.5)
END PROGRAM abstest
% f95 -o abstest abstest.f90
% abstest
 ABS(-2.5) =    2.5000000
 ABS(-2)   =  2
 ABS(-1)   =  1
 ABS( 0)   =  0
 ABS( 1)   =  1
 ABS( 2)   =  2
 ABS( 2.5) =    2.5000000
```

(An interesting property of `ABS`: if its argument is an `INTEGER`, it returns an `INTEGER` result, but if its argument is a `REAL`, it returns a `REAL` result. More on this later in the semester, if we have time.)

Jargon: in programming, the use of a function in an expression is referred to as an *invocation*, or more colloquially as a *call*. We say that the statement

```
    PRINT *, ABS(-2)
```

invokes or *calls* the function `ABS`; the statement *passes* an argument of -2 to the function; the function `ABS` *returns* a value of 2.

Intrinsic Functions in Fortran 90

Fortran 90 has a bunch of functions built into the language. These functions are referred to as *intrinsic* functions, where *intrinsic* means "built into the language." A few examples:

Function name	Math Name	Value	Example		
`ABS(x)`	absolute value	$\|x\|$	`ABS(-1.0)`	$\rightarrow$	1.0
`SQRT(x)`	square root	$x^{1/2}$	`SQRT(2.0)`	$\rightarrow$	1.414...
`EXP(x)`	exponential	e^x	`EXP(1.0)`	$\rightarrow$	2.718...
`LOG(x)`	natural logarithm	$\ln x$	`EXP(2.718...)`	$\rightarrow$	1.0
`LOG10(x)`	common logarithm	$\log x$	`LOG10(100.0)`	$\rightarrow$	2.0
`SIN(x)`	sine	$\sin x$	`SIN(3.14...)`	$\rightarrow$	0.0
`COS(x)`	cosine	$\cos x$	`COS(3.14...)`	$\rightarrow$	-1.0
`TAN(x)`	tangent	$\tan x$	`TAN(3.14...)`	$\rightarrow$	0.0
`CEILING(x)`	least integer $\geq$ `x`	$\lceil x \rceil$	`CEILING(2.5)`	$\rightarrow$	3
`FLOOR(x)`	greatest integer $\leq$ `x`	$\lfloor x \rfloor$	`FLOOR(2.5)`	$\rightarrow$	2
`INT(x)`	`x` truncated toward 0	[x]	`INT(2.5)`	$\rightarrow$	2

You'll find an exhaustive list of all of Fortran 90's intrinsic functions (more than 100 of them) in *Programming in Fortran 90/95*, 1st or 2nd edition, Chapter 17.

As it turns out, the set of intrinsic functions is **grossly insufficient** for most real world tasks, so in Fortran 90, **and in most programming languages**, there are ways for programmers to develop their own *user-defined functions*, which we'll learn more about in a future lesson.

Math: Domain & Range

In mathematics, we refer to the set of numbers that can be the **argument** of a given function as the *domain* of that function.

Similarly, we refer to the set of numbers that can be the **result** of a given function as the *range* of that function.

For example, in the case of the function

$$f(x) \ = \ x \ + \ 1$$

we define the domain to be the set of real numbers (sometimes denoted $\Re$), which means that the x in $f(x)$ can be any real number. Likewise, we also define the range to be the set of real numbers, because for every real number y there is some real number x such that $f(x) \ = \ y$.

On the other hand, for a function

$$q(x) \ = \ \frac{1}{x-1}$$

the domain cannot include 1, because

$$q(1) \ = \ \frac{1}{1-1} \ = \frac{1}{0}$$

which is undefined. So the domain might be $\Re - \{1\}$ (the set of all real numbers except 1).

In that case, the range of q would be the set of all real numbers except 0, because there's no real number y such that $1/y$ is 0.

(Note: if you've taken calculus, you've seen that, as y gets arbitrarily large, $1/y$ approaches 0 as a limit – but "gets arbitrarily large" is not a real number, and neither is "approaches 0 as a limit.")

Programming: Argument Type

Fortran 90 has analogous concepts to the mathematical domain and range: *argument type* and *return type*.

The *argument type*, not surprisingly, is the data type that Fortran 90 expects for an argument of a particular intrinsic function.

Some Fortran 90 intrinsic functions, such as `ABS`, will allow you to pass either an `INTEGER` or a `REAL` as an argument, but you still have to use a numeric data type. If you try to pass, say, a `LOGICAL` value to `ABS`, the compiler gets upset:

```
% cat abslgcarg.f90
PROGRAM abs_logical_argument
    IMPLICIT NONE
    LOGICAL :: lgc = .TRUE.
    PRINT *, ABS(lgc)
END PROGRAM abs_logical_argument
% f95 -o abslgcarg abslgcarg.f90
Error: abslgcarg.f90, line 4:
  Non-numeric argument to
  numeric intrinsic ABS
[f95 error termination]
```

Programming: Return Type

Just as the programming concept of argument type is analogous to the mathematical concept of domain, so too is the programming concept of return type analogous to the mathematical concept of range.

The *return type* of a Fortran 90 function is the data type of the value that the function returns. The return value is **guaranteed** to have that data type, and the compiler gets upset if you use the return value inappropriately:

```
% cat abslgcret.f90
PROGRAM abs_logical_return
    IMPLICIT NONE
    LOGICAL :: lgc
    lgc = ABS(-2.0)
END PROGRAM abs_logical_return
% f95 -o abslgcret abslgcret.f90
Error: abslgcret.f90, line 4:
  Incompatible data types
  in assignment statement
[f95 error termination]
```

More on Function Arguments

In mathematics, a function argument can be:

- a number:
  $f(5) \;=\; 5 \;+\; 1 \;=\; 6$
- a variable:
  $f(z) \;=\; z \;+\; 1$
- an arithmetic expression:
  $f(5+7) \;=\; (5+7) \;+\; 1 \;=\; 12 \;+\; 1 \;=\; 13$
- another function:
  $f(a(w)) \;=\; |w| \;+\; 1$
- any combination of these; i.e., any general expression whose value is in the domain of the function:
  $f(3a(5w+7)) \;=\; 3\,(|5w+7|) \;+\; 1$

Likewise, in Fortran 90 the argument of a function can be any non-empty expression **that evaluates to an appropriate data type**, including an expression containing a function call.

Function Argument Example

```
% cat funcargs.f90
PROGRAM funcargs
    IMPLICIT NONE
    REAL,PARAMETER :: pi = 3.1415926
    REAL :: angle_in_radians
    PRINT "(A,F9.7,A,F10.7)",                               &
 &     "COS(", 1.5707963, ") = ", COS(1.5707963)
    PRINT "(A,F9.7,A,F10.7)", "COS(", pi, ") = ",    &
 &      COS(pi)
    PRINT "(A)", "Enter an angle in radians:"
    READ *, angle_in_radians
    PRINT "(A,F10.7,A,F10.7)",                              &
 &      "COS(", angle_in_radians, ") = ",                   &
 &      COS(angle_in_radians)
    PRINT "(A,F10.7,A,F10.7)",                              &
 &      "ABS(COS(", angle_in_radians, ")) = ",       &
 &     ABS(COS(angle_in_radians))
    PRINT "(A,F10.7,A,F10.7)",                              &
 &      "COS(ABS(", angle_in_radians, ")) = ",       &
 &     COS(ABS(angle_in_radians))
    PRINT "(A,F10.7,A,F10.7)",                              &
 &      "ABS(COS(2.0 * ", angle_in_radians, ")) = ", &
 &      ABS(COS(2.0 * angle_in_radians))
    PRINT "(A,F10.7,A,F10.7)",                              &
 &      "ABS(2.0 * COS(", angle_in_radians, ")) = ", &
 &      ABS(2.0 * COS(angle_in_radians))
    PRINT "(A,F10.7,A,F10.7)",                              &
 &      "ABS(2.0 * COS(1.0 / 5.0 * ",                &
 &      angle_in_radians, ")) = ",                   &
 &      ABS(2.0 * COS(1.0 / 5.0 * angle_in_radians))
END PROGRAM funcargs
% f95 -o funcargs funcargs.f90
% funcargs
COS(1.5707963) =  0.0000001
COS(3.1415925) = -1.0000000
Enter an angle in radians:
 -3.1415926
COS(-3.1415925) = -1.0000000
ABS(COS(-3.1415925)) =  1.0000000
COS(ABS(-3.1415925)) = -1.0000000
ABS(COS(2.0 * -3.1415925)) =  1.0000000
ABS(2.0 * COS(-3.1415925)) =  2.0000000
ABS(2.0 * COS(1.0 / 5.0 * -3.1415925)) =  1.6180340
```

Using Functions

Functions are used **in expressions** in exactly the same ways that variables and constants are used. For example, a function call can be used on the right hand side of an assignment:

```
    REAL :: theta = 3.1415926 / 4.0
    REAL :: cos_theta
    cos_theta = COS(theta)
```

A function call can also be used in an expression in list-directed output:

```
    PRINT *, 2.0
    PRINT *, COS(theta) ** 2.0
```

And, since any expression can be used as some function's argument, a function call can also be used as an argument to another function:

```
    REAL,PARAMETER :: pi = 3.1415926
    PRINT *, 1 + COS(ASIN(SQRT(2.0)/2.0) + pi)
```

However, most function calls cannot be used in initialization:

```
! * ILLEGAL ILLEGAL ILLEGAL
    REAL :: cos_theta = COS(3.1415926)
! * ILLEGAL ILLEGAL ILLEGAL
```

Nor can most function calls be used in named constant declaration:

```
! * ILLEGAL ILLEGAL ILLEGAL
    REAL,PARAMETER :: cos_theta = COS(3.1415926)
! * ILLEGAL ILLEGAL ILLEGAL
```

As a general rule, it's best not to try to use function calls in initializations or in named constant declarations.

Function Use Example

```
% cat funcuse.f90
PROGRAM function_use
    IMPLICIT NONE
    REAL,PARAMETER :: pi = 3.1415926
!   REAL,PARAMETER :: sin_pi = SIN(3.1415926) ! ILLEGAL
!   REAL :: sin_pi = SIN(pi)                  ! ILLEGAL
!   REAL :: sin_pi = SIN(3.1415926)           ! ILLEGAL
    REAL :: theta, sin_pi, sin_theta
    theta     = 3.0 * pi / 4
    sin_pi    = SIN(pi)
    sin_theta = SIN(theta)
    PRINT *, "2.0        = ", 2.0
    PRINT *, "pi         = ", pi
    PRINT *, "theta      = ", theta
    PRINT *, "SIN(pi)    = ", SIN(pi)
    PRINT *, "sin_pi     = ", sin_pi
    PRINT *, "SIN(theta) = ", SIN(theta)
    PRINT *, "sin_theta  = ", sin_theta
    PRINT *, "SIN(theta) ** (1.0/3.0)  = ", &
 &           SIN(theta) ** (1.0/3.0)
    PRINT *, "1 + SIN(ACOS(1.0))       = ", &
 &           1 + SIN(ACOS(1.0))
    PRINT *, "SIN(ACOS(1.0))           = ", SIN(ACOS(1.0))
    PRINT *, "SQRT(2.0)                = ", SQRT(2.0)
    PRINT *, "SQRT(2.0) / 2            = ", SQRT(2.0) / 2
    PRINT *, "ACOS(SQRT(2.0)/2.0)      = ", &
 &           ACOS(SQRT(2.0)/2.0)
    PRINT *, "SIN(ACOS(SQRT(2.0)/2.0)) = ", &
 &           SIN(ACOS(SQRT(2.0)/2.0))
END PROGRAM function_use
% f95 -o funcuse funcuse.f90
% funcuse
 2.0        =    2.0000000
 pi         =    3.1415925
 theta      =    2.3561945
 SIN(pi)    =    1.5099580E-07
 sin_pi     =    1.5099580E-07
 SIN(theta) =    0.7071068
 sin_theta  =    0.7071068
 SIN(theta) ** (1.0/3.0)  =    0.8908987
 1 + SIN(ACOS(1.0))       =    1.0000000
 SIN(ACOS(1.0))           =    0.0000000E+00
 SQRT(2.0)                =    1.4142135
 SQRT(2.0) / 2            =    0.7071068
 ACOS(SQRT(2.0)/2.0)      =    0.7853982
 SIN(ACOS(SQRT(2.0)/2.0)) =    0.7071068
```

Evaluation of Functions in Expressions

When a function call appears in an expression – for example, on the right hand side of an assignment statement – the function is *evaluated* just before its value is needed, in accordance with the rules of precedence order.

For example, if `x` and `y` are `REAL` variables, and `y` has already been assigned the value -10.0, then the assignment statement

```
x = 1 + 2.0 * 8.0 + ABS(y) / 4.0
```

is evaluated like so:

```
x =  1  + 2.0 * 8.0 +   ABS(y)   / 4.0 ⇒

x =  1  +    16.0   +   ABS(y)   / 4.0 ⇒

x =  1  +    16.0   + ABS(-10.0) / 4.0 ⇒

x =  1  +    16.0   +    10.0    / 4.0 ⇒

x =  1  +    16.0   +        2.5       ⇒

x = 1.0 +    16.0   +        2.5       ⇒

x =        17.0     +        2.5       ⇒

x =                   19.5
```

So, the variable `x` is ultimately assigned the value 19.5.

Functions with Multiple Arguments

In mathematics, we sometimes have functions that have multiple arguments:

$$h(x, y) = xy + 2x + 3y + 5$$

In this case, we know that

$$\begin{array}{lclclclclcl}
h(-2.5, -1.5) & = & (-2.5)(-1.5) & + & (2)(-2.5) & + & (3)(-1.5) & + & 5 & = & -0.75 \\
h(-2, -0.5) & = & (-2)(-0.5) & + & (2)(-2) & + & (3)(-0.5) & + & 5 & = & +0.5 \\
h(-1, 1.25) & = & (-1)(1.25) & + & (2)(-1) & + & (3)(1.25) & + & 5 & = & 0 \\
h(0, 0) & = & (0)(0) & + & (2)(0) & + & (3)(0) & + & 5 & = & +5
\end{array}$$

Here, we define the domain of the first argument of the function h to be the set of all real numbers $\Re$, and likewise we define the domain of the second argument of h to be the set of all real numbers $\Re$, so the domain of h as a whole is $\Re \times \Re$, pronounced "real times real."

Since the result of h is a single real value, we define the range of h to be $\Re$, and we denote the mapping as

$$h : \Re \times \Re \mapsto \Re$$

Similarly, in Fortran 90 we have intrinsic functions with multiple arguments. Examples include:

Function name	Math Name	Math Value
`MIN(x,y)`	minimum	if (`x` $<$ `y`) then `x` else `y`
`MAX(x,y)`	maximum	if (`x` $>$ `y`) then `x` else `y`
`MOD(x,y)`	remainder	`x - (INT(x / y) * y)`
`BTEST(x,y)`	bit test	`.TRUE.` if the `y`th bit of `x` is 1

Functions with multiple arguments can be used in exactly the same ways as functions with a single argument.

In a function call, the list of arguments must be in **exactly** the correct order and have **exactly** the correct data types.