Subroutines and Functions
Procedures: Subroutines and Functions

There are two types of procedures:

- **SUBROUTINE**: a parameterized named sequence of code which performs a specific task and can be invoked from other program units.
  - invoked with the `CALL` statement
- **FUNCTION**: same as a subroutine but returns a result in the function name.
  - invoked by placing the function name (and its associated arguments in an expression)
  - use when just one return value is needed.
- Example: `sort3.f90` and `sort_3.f90`
This simple example illustrates one of the important uses of subroutines: To exhibit the overall structure of a program and put the details in another place.

Internal subroutines and functions are designated by the `contains` statement.

The `implicit none` in the host program applies to the internal subroutines. Also used in modules.

Can we go even further with this example?

Look at `sort_3a.f90`
Subroutines with Arguments

- We can pass values to a subroutine by placing them in parentheses after the name of the subroutine in the call statement.
- In the call to swap, n1 and n2 are called arguments.
- To make subroutine swap available to other program units, we would need to place it within a module.
Functions

* Just like a subroutine, but intended to return one value (or an array of values). Invoked just like an intrinsic function.

* The result of a function should be placed in a result variable using the `result` keyword at the end of the function statement.

* If the result keyword and variable are omitted, the function name is used as the return variable and must be declared in the function.

* Example: `series.f90`
Argument Association

The variables $a$ and $b$ in subroutine swap are placeholders for the two numbers to be swapped. These are dummy arguments and must be declared in the subroutine. The variables $n1$ and $n2$ in the first call to swap are the actual arguments.

If the value of a dummy argument changes, then so does the value of the actual argument (pass-by-reference).

An actual argument that is a constant or an expression more complicated than a variable can only pass a value to the corresponding dummy argument. This is called pass-by-value.
It is bad programming practice to modify arguments in function calls.

In general, the number of actual and dummy arguments must be the same.

Also, the data type (and kind parameter) of each actual argument must match that of the corresponding dummy argument.

Keyword arguments and optional arguments: best explained by an example (series2.f90)
Argument Intent

It’s good programming practice to indicate whether a dummy argument will be:

- Only be referenced -- **INTENT(IN)**
- Be assigned to before use -- **INTENT(OUT)**
- Be referenced and assigned to -- **INTENT(INOUT)**

The use of INTENT attributes is recommended as it:

- Allows good compilers to check for coding errors.
- Facilitates efficient compilation and optimization.

**Example:** series3.f90
Scope

* The scope of a name is the set of lines in a Fortran program where that name may be used and refer to the same variable, procedure or type.

* In general, the scope extends throughout the program unit where the entity is declared (host association).
  - Known to any procedures declared within.
  - **Example:** calculatepay.f90

* But **NOT** if the same entity is redeclared in an internal procedure. (**myscope.f90**)

* Module scope is a little different -- we’ll cover that later (**use association**).
The Save Attribute

Fortran 77 compilers generally used static storage for all variables. Most Fortran 90 systems use static storage only when required. This means that local variables in subroutines and functions will NOT be preserved after control returns unless:

- The variable is initialized.
- The **SAVE** attribute is used.

There’s also a SAVE statement but the use of the attribute in declarations is the preferred usage.

```
real, save :: p, q
```
The Return Statement

- **RETURN** causes execution of a procedure to terminate with control given back to the calling program.
- Can be useful in more elaborate procedures as an alternative to a complicated set of nested if constructs.