OUTLINE

1 REVIEW

2 FUNCTIONS
   - Why use functions
   - How do they work

3 SUBROUTINES
   - Why use subroutines?
   - How do they work

4 FLOW CONTROL
   - Logical Control
   - Looping

5 SUMMARY
# Outline

<table>
<thead>
<tr>
<th>Review</th>
<th>Functions</th>
<th>Subroutines</th>
<th>Flow Control</th>
<th>Summary</th>
</tr>
</thead>
</table>

1. **Review**

2. **Functions**
   - Why use functions
   - How do they work

3. **Subroutines**
   - Why use subroutines?
   - How do they work

4. **Flow Control**
   - Logical Control
   - Looping

5. **Summary**
<table>
<thead>
<tr>
<th>Outline</th>
<th>Review</th>
<th>Functions</th>
<th>Subroutines</th>
<th>Flow Control</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Functions</td>
<td>Why use functions</td>
<td>How do they work</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Subroutines</td>
<td>Why use subroutines?</td>
<td>How do they work</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Flow Control</td>
<td>Logical Control</td>
<td>Looping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Summary</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OUTLINE

1. REVIEW
2. FUNCTIONS
   - Why use functions
   - How do they work
3. SUBROUTINES
   - Why use subroutines?
   - How do they work
4. FLOW CONTROL
   - Logical Control
   - Looping
5. SUMMARY
OUTLINE

1 REVIEW

2 FUNCTIONS
   • Why use functions
   • How do they work

3 SUBROUTINES
   • Why use subroutines?
   • How do they work

4 FLOW CONTROL
   • Logical Control
   • Looping

5 SUMMARY
**THE STORY SO FAR...**

- Understand a little bit about computers
- Understand about data types and how to assign them
- Be able to input data from keyboard...
- and output to screen
- Should be ready to write first programs...
THE STORY SO FAR...

- Understand a little bit about computers
- Understand about data types and how to assign them
- Be able to input data from keyboard...
- and output to screen
- Should be ready to write first programs...
THE STORY SO FAR...

- Understand a little bit about computers
- Understand about data types and how to assign them
- Be able to input data from keyboard...
- and output to screen
- Should be ready to write first programs...
**OUTLINE**

1. **REVIEW**

2. **FUNCTIONS**
   - Why use functions
   - How do they work

3. **SUBROUTINES**
   - Why use subroutines?
   - How do they work

4. **FLOW CONTROL**
   - Logical Control
   - Looping

5. **SUMMARY**
Why Functions?

- Functions can be used to break up the code into smaller pieces
- Don’t want to write the same piece of code twice
- Can be tested on its own
- Fortran has many intrinsic functions
- Libraries can provide us with special functions
**Why Functions?**

- Functions can be used to break up the code into smaller pieces
- Don’t want to write the same piece of code twice
- Can be tested on its own
- Fortran has many intrinsic functions
- Libraries can provide us with special functions
Why Functions?

- Functions can be used to break up the code into smaller pieces
- Don’t want to write the same piece of code twice
- Can be tested on its own
- Fortran has many intrinsic functions
- Libraries can provide us with special functions
WHAT DO THEY DO?

- Functions have a `return` value
- Take in variables as arguments
- Written in the code like a mathematical function

**INTRINSIC AND EXTERNAL FUNCTIONS**

```plaintext
! Intrinsic functions in action
y = sin(x) * exp(-x/2.)
! Our own function
z = my_func(x) * another_func(x)
```
WHAT DO THEY DO?

- Functions have a **return** value
- Take in variables as arguments
- Written in the code like a mathematical function

**INTRINSIC AND EXTERNAL FUNCTIONS**

> ! Intrinsic functions in action
> \[ y = \sin(x) \times \exp(-x/2.) \]
> ! Our own function
> \[ z = \text{my\_func}(x) \times \text{another\_func}(x) \]
## Outline

1. **Review**
2. **Functions**
   - Why use functions
   - How do they work
3. **Subroutines**
   - Why use subroutines?
   - How do they work
4. **Flow Control**
   - Logical Control
   - Looping
5. **Summary**
We write functions underneath the main program.

General form is:

```fortran
FUNCTION my_func(x,y)
    ! Need to declare type of my_func
    REAL my_func
    ! Variable declarations
    REAL x,y
    ! Local variables
    INTEGER i,j
    ! Executable statements
    my_func = ...
END FUNCTION
```
WHAT TO TELL THE COMPILER

- Compiler needs to know the ‘return’ type of function
- Must declare any *external* functions used inside a program
- Inside your program:
  
  ! Declare your functions with TYPE,EXTERNAL
  REAL, EXTERNAL :: my_func

- Functions can be used inside other functions and subroutines
# OUTLINE

1. **Review**
2. **Functions**
   - Why use functions
   - How do they work
3. **Subroutines**
   - Why use subroutines?
   - How do they work
4. **Flow Control**
   - Logical Control
   - Looping
5. **Summary**
They act like algorithms to split up a program
Write generic routines they can be used in different programs
Use when several different variables change value
Can be tested on its own
Libraries can provide special routines
**USING SUBROUTINES**

- They act like algorithms to split up a program
- Write generic routines they can be used in different programs
- Use when several different variables change value
- Can be tested on its own
  - Libraries can provide special routines
**Using subroutines**

- They act like algorithms to split up a program.
- Write generic routines they can be used in different programs.
- Use when several different variables change value.
- Can be tested on its own.
- Libraries can provide special routines.
**WHAT DO THEY DO?**

- No return value
- Can pass variables in and out

**SUBROUTINES**

! Call a subroutine
CALL my_routine(x,y)

! Call another subroutine
CALL another_routine(x,y,z,string)
WHAT DO THEY DO?

- No return value
- Can pass variables in and out

**SUBROUTINES**

! Call a subroutine
CALL my_routine(x,y)

! Call another subroutine
CALL another_routine(x,y,z,string)
# OUTLINE

## 1. REVIEW

## 2. FUNCTIONS
- Why use functions
- How do they work

## 3. SUBROUTINES
- Why use subroutines?
- How do they work

## 4. FLOW CONTROL
- Logical Control
- Looping

## 5. SUMMARY
Writing a Subroutine

- We write subroutines underneath the main program.
- Don’t need to be declared in program since there is no return value.
- General form is:

```fortran
SUBROUTINE my_routine(x,y)
  ! Variable declarations
  REAL, INTENT(INOUT) x,y
  ! Local variables
  INTEGER i,j
  ! Executable statements
  x = ...
END SUBROUTINE
```
WRITING A SUBROUTINE

- We use the **INTENT** keyword to specify what we want to do with variables

**REAL, INTENT(IN) :: x**
x is real, has a value coming into the routine. Not to be changed inside.

**REAL, INTENT(OUT) :: x**
x is real, **NOT** assigned a value coming into the routine. Assigned a value inside the routine.

**REAL, INTENT(INOUT) :: x**
x is real, has a value coming into the routine. Can be changed inside.
**WRITING A SUBROUTINE**

- We use the `INTENT` keyword to specify what we want to do with variables.

```plaintext
REAL, INTENT(IN) :: x
x is real, has a value coming into the routine. Not to be changed inside.

REAL, INTENT(OUT) :: x
x is real, **NOT** assigned a value coming into the routine. Assigned a value inside the routine.

REAL, INTENT(INOUT) :: x
x is real, has a value coming into the routine. Can be changed inside.
```
WRITING A SUBROUTINE

- We use the **INTENT** keyword to specify what we want to do with variables

```plaintext
REAL, INTENT(IN) :: x
x is real, has a value coming into the routine. Not to be changed inside
```

```plaintext
REAL, INTENT(OUT) :: x
x is real, **NOT** assigned a value coming into the routine. Assigned a value inside the routine
```

```plaintext
REAL, INTENT(INOUT) :: x
x is real, has a value coming into the routine. Can be changed inside
```
WRITING A SUBROUTINE

- We use the **INTENT** keyword to specify what we want to do with variables

**REAL, INTENT(IN) :: x**

x is real, has a value coming into the routine. Not to be changed inside.

**REAL, INTENT(OUT) :: x**

x is real, **NOT** assigned a value coming into the routine. Assigned a value inside the routine.

**REAL, INTENT(INOUT) :: x**

x is real, has a value coming into the routine. Can be changed inside.
OUTLINE

1 REVIEW

2 FUNCTIONS
   • Why use functions
   • How do they work

3 SUBROUTINES
   • Why use subroutines?
   • How do they work

4 FLOW CONTROL
   • Logical Control
   • Looping

5 SUMMARY
Aside adding and multiplying, a computer can also check whether a statement is true or false

**DECLARING A LOGICAL VARIABLE**

```
LOGICAL :: the_truth, a_lie
the_truth = .TRUE.
a_lie = .FALSE.
```

Can be combined with other variables using `.NOT.`, `.AND.` and `.OR.`

**TWO WRONGS MAKE A RIGHT**

```
LOGICAL :: wrong_1, wrong_2, a_right
wrong_1 = .FALSE. ; wrong_2 = .FALSE.
a_right = (.NOT. wrong_1) .AND. (.NOT. wrong_2)
```
Logical Statements

Aside adding and multiplying, a computer can also check whether a statement is true or false.

Declaring a Logical Variable

```fortran
LOGICAL :: the_truth, a_lie
the_truth = .TRUE.
a_lie = .FALSE.
```

Can be combined with other variables using .NOT., .AND. and .OR.

Two Wrongs Make a Right

```fortran
LOGICAL :: wrong_1, wrong_2, a_right
wrong_1 = .FALSE. ; wrong_2 = .FALSE.
a_right = (.NOT. wrong_1) .AND. (.NOT. wrong_2)
```
Aside adding and multiplying, a computer can also check whether a statement is true or false.

**DECLARING A LOGICAL VARIABLE**

```fortran
LOGICAL :: the_truth, a_lie
the_truth = .TRUE.
a_lie = .FALSE.
```

- Can be combined with other variables using `.NOT.`, `.AND.`, and `.OR.`

**TWO WRONGS MAKE A RIGHT**

```fortran
LOGICAL :: wrong_1, wrong_2, a_right
wrong_1 = .FALSE. ; wrong_2 = .FALSE.
a_right = (.NOT. wrong_1) .AND. (.NOT. wrong_2)
```
## Logical Statements

- Aside adding and multiplying, a computer can also check whether a statement is true or false

### Declaring a Logical Variable

<table>
<thead>
<tr>
<th>LOGICAL</th>
<th>the_truth, a_lie</th>
</tr>
</thead>
<tbody>
<tr>
<td>the_truth</td>
<td>.TRUE.</td>
</tr>
<tr>
<td>a_lie</td>
<td>.FALSE.</td>
</tr>
</tbody>
</table>

- Can be combined with other variables using **.NOT.**, **.AND.**, and **.OR.**

### Two Wrongs Make a Right

<table>
<thead>
<tr>
<th>LOGICAL</th>
<th>wrong_1, wrong_2, a_right</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrong_1</td>
<td>.FALSE. ; wrong_2 = .FALSE.</td>
</tr>
<tr>
<td>a_right</td>
<td>(.NOT. wrong_1) .AND. (.NOT. wrong_2)</td>
</tr>
</tbody>
</table>
A computer can evaluate the relation between two values

- $a < b$ \texttt{a .LT. b} ! true if $a < b$
- $a > b$ \texttt{a .GT. b} ! true if $a > b$
- $a \leq b$ \texttt{a .LE. b} ! true if $a < b$ or $a = b$
- $a \geq b$ \texttt{a .GE. b} ! true if $a > b$ or $a = b$
- $a == b$ \texttt{a .LT. b} ! true if $a = b$
- $a /= b$ \texttt{a .LT. b} ! true if $a$ doesn’t equal $b$

Be careful when evaluating whether two real numbers are equal!
IF STATEMENTS

- Execute a set of statements if a statement is true

**IF STATEMENT**

```
IF(is_it_true) THEN
  ! Execute statements...
  x = ...
ENDIF
```

- Can also write it as...

```
IF(is_it_true)x=...  ! all on one line
```

- ...if we only have one statement to execute
**IF STATEMENTS**

- Execute a set of statements if a statement is true

**IF STATEMENT**

```plaintext
IF(is_it_true) THEN
    ! Execute statements...
    x = ...
ENDIF
```

- Can also write it as...

```plaintext
IF(is_it_true)x=... ! all on one line
```

- ...if we only have one statement to execute
A COMPLEX IF STATEMENT

REAL :: a,b,c,deter
! Calculate roots of quadratic
READ *, a,b,c
deter = b*b - 4.*a*c ! Calculate determinant
IF(deter>0.) THEN
   PRINT *, "Two roots ::",
   " x_1 =", (-b + SQRT(d))/(2.*a) &
   " x_2 =", (-b - SQRT(d))/(2.*a)
ELSE IF(deter==0.) THEN
   PRINT *, "One double root ::",
   " x_0 =", -b/(2.*a)
ELSE
   PRINT *, "Complex roots"
ENDIF
OUTLINE

1 REVIEW

2 FUNCTIONS
   • Why use functions
   • How do they work

3 SUBROUTINES
   • Why use subroutines?
   • How do they work

4 FLOW CONTROL
   • Logical Control
   • Looping

5 SUMMARY
The ability to perform a task again and again is one of the major advantages of using a computer.

Computers will not complain at doing the same thing over and over.

The task is performed in precisely the same way each time.

Fortran uses the **DO** loop to achieve this.

```fortran
x = 0. ; dx = 0.1
DO i=1,10
  ! Do something ten times
  x = x + dx
END DO
PRINT *,x
```

What is the value of \( x \)?
The ability to perform a task again and again is one of the major advantages of using a computer.

Computers will not complain at doing the same thing over and over.

The task is performed in precisely the same way each time.

Fortran uses the **DO** loop to achieve this.

```fortran
x = 0. ; dx = 0.1
DO i=1,10
  ! Do something ten times
  x = x + dx
END DO
PRINT *,x
```

What is the value of \( x \)?
The generic **DO** statement:

```plaintext
DO i = start, finish, incr
   ! if i<=finish do something
   x = ...
END DO ! i=i+incr and go to DO
```

- The value of `i` can be used but not changed inside the loop.
- The loop is executed if `i<=finish`.
- The value of `i` is increased by `incr` at the end of the loop.
- `start, finish` or `incr` may be a function or expression so long as they return the correct data type.
More control over looping

- We often wish to exit the loop early once some criterion is met.
- To do this we use the keyword **EXIT**

**Example exit loop**

```fortran
DO i=1,max_iters
    ! do some stuff here...
    IF(error<tol) EXIT
END DO
```

- Use indenting to help you keep track of loops.
- Can also name loops if it is becoming difficult:

```fortran
my_loop: DO i = 1,10
    ! more DO loops in here
END DO my_loop
```
**More control over looping**

- We often wish to exit the loop early once some criterion is met.
- To do this we use the keyword `EXIT`.

**Example exit loop**

```
DO i=1,max_iters
  ! do some stuff here...
  IF (error<tol) EXIT
END DO
```

- Use indenting to help you keep track of loops.
- Can also name loops if it is becoming difficult:

```
my_loop: DO i = 1,10
  ! more DO loops in here
END DO my_loop
```
# In This Lecture

- How to declare, write and use Functions
- How to declare, write and use Subroutines
- Understand logical statements
- Use logic to control the flow of your program
- Use loops to repeat parts of your program