Fortran Commands

- FORTRAN = Formula TRANslator
- gfortran will compile Fortran 77 or 90/95
  use file extension .f for fixed format (column 7.)
  " " " .f90 " free format

- implicit none – means all variable must be declared
- (Kind=8) means 8 bytes used for storage
- 3.d0 ⇒ 3×10^0 in double Precision (8 bytes)
  2.d-1 ⇒ $2 \times 10^{-1} = 0.2$
- Print*, "z = ", z * means no special format specified.

- To give different name with -o flag

```
$ gfortran example.f90 -o example1.exe
$ ./example1.exe
 z = 3.2000
```

→ What does 'Implicit none' actually do?

```
program example1
    implicit none
    real (Kind =8) :: x, y, z
    x = 3.d0
    y = 2.d-1
error ⟶ zz = x + y
end program example1
```

Here we get error: Symbol 'zz' at (1) has no Implicit type

```
Program example1
    real (Kind=8) :: x, y, z
    x = 3.d0
    y = 2.d-1
    zz = x + y        → no error
end program example1
```

Here we do not get error

→ Variable refer to particular storage location(s), must declare variable to be of a particular type and this won't change

The statement
implicit none
means all variable must be explicitly declared.

Otherwise you can use a variable without prior declaration and the type will depend on what letter the name starts with.

- Default:
  - integer if start with i, j, k, l, m, n
  - real (kind = 4) otherwise (single precision)

Fortran loops and arrays

```
! loop1.f90
Program loop1
    implicit none
    integer, parameter :: n = 1000          ← integer, parameter means this value will not be changed
    real (kind=8), dimension(n) :: x, y     ← means these are arrays of dimension/length n
    integer :: i

    do i = 1, n
x(i) = 3.d0 * i      ← ith element of (x)
    enddo

    do i = 1, n
        y(i) = 2.d0 * x(i)
    enddo
    Print *, "Last y computed: ", y(n)

end program loop1
```

Fortran if then else

```
! ifelse.f90
Program ifelse1
    implicit none
    real (kind=8) :: x
    integer :: i
    i = 3
    if (i <= 2) then
        Print *, "i is less or equal to 2"
    else if (i /= 5) then
        Print *, "i is greater than 2, and equal to 5"
    else Print *, "i is equal to 5"
    end if → end program ifelse1
```

Booleans: .true. .false.

Comparisons:

< or .lt.	<= or .le.	> or .gt.
>= or .ge.	== or .eq.	/= or .ne.

Example

```
if ((i >= 5) .and. (i <12)) then
if (((i .lt. 5) .or. (i .ge. 12)) .and. &
    (i .ne. 20)) then
```

& → fortran continuation character

→ Function in Fortran

- Function take some input arguments and return a single value.

  Usage: $y = f(x)$ or $z = g(x, y)$

- Should be declared as external with the type of value returned

```
real (kind = 8), external :: f
```

- real (kind = 8), intent(in) :: x → here intent me that x is only input to the function, do not modify x in the function

→ Subroutines → have arguments each of which might be for input or output or both

Usage: call sub1 (x, y, a, z, b)

(on specify the intent of each argument as)

```
real (kind=8), intent(in) :: x, y
  "     "    "   "   (out) :: z
  "     "    "   "   (inout):: a, b
```

that specifies that a, b are passed in and may be modified x, y are passed in and not modified and z may not have value coming in but will be set by sub1.

Operation on Array in Fortran

Print *, "x + y = "
" " x * y = (1 = (x(1)y(1), x(2)y(2), ..)

Print *, "sqrt(y) = " → square component wise

Print *, "dot_product(x,y) = " → scalar product component wise

- Fortran Reshape fills array by columns

A = reshape ((/1,2,3,4,5,6/), (/3,2/))

$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$ in python it would be $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

- c = matmul(a,b) → multiply matrix a and b

linear system in Fortran

We have to use LAPACK - linear algebra package to solve linear system. We can't do (c = a\b) in fortran.

Note → real(kind=8) dimension(:), allocate :: x
" " " " (:,:), " :: a

allocate command helps to specify the dimension after defining or in between the program

allocate (x(10))
" (a(30,10))

! use arrays
! then clean up
deallocate (x)
" (a)

- Some time the allocation does not happen due to insufficient memory so to check it we can use 'stat' function

allocate (a(30000,10000), stat = alloc_error)
if (alloc_error /= 0) then
Print *, "Insufficient memory"

- In fortran gdb is the debugger. Use 'help' command to get all the commanding gdb debugger when it is active

Makefiles

- Makefiles give a way to recompile only parts of the code that have changed.
- Also used for checking dependancies in other build systems e.g creating figures, running data, building etc to construct a manuscript.

→ To run multiple files of fortran:

$ gfortran main.f90 sub1.f90 sub2.f90, \-o name

(main.f90 sub1.f90 sub2.f90: name of file; name: name of output file)

Then run executable

$ ./main.exe

→ We can run & compile file separately:

$ gfortran -c main.f90 sub1.f90 sub2.f90

(-c: To compile; main.f90 sub1.f90 sub2.f90: name of file)

Then:

$ gfortran main.o sub1.o sub2.o -o main.exe

$ ./main.exe > output.txt

→ Advantage of this is that if now we modify sub2.f90, we only need to recompile this file

$ gfortran -c sub2.f90

$ " main.o sub1.o sub2.o -o main.exe

$ ./main.exe

- But when we have 1000 of file it will become tedious task so, we use makefiles for that.
- To run makefile use can keep name of the such as 'Makefile' or 'Makefile_1' and format can be .txt or anything.

→ To run just type "make main.exe"

Target file name that you want to execute

→ Typical element in the simple Makefile

target: dependencies

<TAB> command(s) to make target

e.g no output.txt : main.exe

[TAB]./main.exe > output.txt

command

output.txt → target

main.exe → dependencies

→ Typing "make target" means:

① Make sure all the dependencies are up to date uptodate means the moment the last time it was compile, is any change made to it from that time to at present.

② If target is older than any dependency recreate it using the specified commands.

→ To check the last time the file is created and was modified type (ls -l filename.*)

→ "touch filename" it just the file update the file name date and time it created.
If no file of that name exist it will create the file with that name.

→ Replace the pattern rule, use which help to reduce the code length and make it easier when have many file with same extension. ⑪

"%.o : %.f90
gfortran -c $<" Instead of 'main.o : main.f90 gfortran -c main.f90'

→ To use new make file we have to type
→ 'make -f Newfile name'
↳ f means force

→ 'make main.exe -f newfile name'
We also use this to run makefile with name other than 'Makefile'

→ If we have large number of files, code and we have to write them all number of time, so to save time we can use $(variable name) = name all file names
When we have to use to, we have to write "$(variable name)".

→ FC ?= gfortran → This will run the code with default compiler if no gfortran name compiler is present in system

→ To remove the dependencies file write
.PHONY : clean

clean :
rm -f $(OBJECTS) main.exe
↳ File name to be removed.

→ Common makefile error are, Use spaces instead of instead of TAB, we get error.
"Makefile:5: 14: * * * missing separator. Stop."

1s on how many digit
bit = "binary digit"

→ @ echo → means print out the string not commands

· If I add '@' in front of any commands it will seen silently without showing its output.

- It depends on how many bytes used for each real number.
  1 byte = 8 bits bit = "binary digit"

A $10,000 \times 10,000$ matrix = 10^8 elements

So required 8×10^8 bytes = 800 MB

Memory is divided bytes, consisting of 8 bits each. So one byte can hold $2^8 = 256$ distinct bit patterns

0 — 255

256 numbers

Slide-12]

- To convert (13)$_{10}$ in Binary Form.

$2 \overline{)13}\ 6$ r 1
$2 \overline{)6}\ 3$ r 0
$2 \overline{)3}\ 1$ r 1
$2 \overline{)1}\ 0$ r 1

(↓)

- Whenever divide last number with 0 quotient we go in reverse other (1101) (↓)

2) 13 (6
12
1

2) 6 (3
6
0

2) 3 (1
2
1

2) 1 (0
0
1

00000

2) 10 (5
10
0

1101 — 13_{10}

- Considering 8 digit and zero in place of other places ahead it 00001101 (↓)

00001101
11110010 + 1
11110011

- now reverse the order and add 1

11110010 + 1 = **11110011**

- if we have 2 (↓) at the end we then binary of 2 is 10 and carry 10 forward

- To clone a reporisitory we use;
  $ git clone \ https://bitbucket.org/orjdave/workspace
  \ name of folder / file
- To count the number of file and directory in a given folder is 'ls | wc -l' this will give you total number of files + dire in a folder.
  - ls -l|wc-l → will give one extra number with the because it have one line giving total count
  - tree command directly give count of number and directories and file
- To for generate new file we have following commands such as;
  - cat > file name
  - touch file name
  - > filename
  - echo >> filename

→ To add text to a file, we have
  - cat >> file name
    ==== text
    ctrl D

→ (I) echo -e 'This will be the text /n for new line' >> filename

→

Loops in shell

Syntax:

```
for thing in list-of-things
do
    operation-using $thing
done
```

cat > test filename → will delete the text from store in the file.

cat >> test filename → will concatenate the data in the file without deleting the from one.

→ to save data in the file

```
for things in *.pdb
do
    echo $things          → only print file name
    cat $thing >> thing.pdb → concatenate the data of file in thing.pdb
done
```

→ Exercise - bash

```
for thing in *.pdb
do
    echo $thing
    cat $thing | head -1 >> test.txt
    $this | tail -2 | head -1
done
```

cat ($the >>

cp

→ for thing in *.pdb

```
do
    echo $things
    cp $thing
```

PYTHON Exercise

range(N) → [0, 1, 2, ..., N-1]

Newton method

~~f(x)~~ $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$ or $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

→ $f(s) = s^2 - x = 0$ (For square root)

$s = s - \frac{2s}{s^2 - x}$ $\quad s_2 = s_1 - \frac{s^2 - x}{2s}$

$\leftarrow \quad s_2 = s_1 - \frac{s}{2} + \frac{x}{2s}$

$s_2 = s_1 - \frac{s_1}{2} + \frac{x}{2s}$

$s_2 = \frac{s_1}{2} + \frac{x}{2s_1}$

$$s_2 = \frac{1}{2}\left(s_1 + \frac{x}{s_1}\right)$$

→ To install some of the fel use

$ sudo apt-get install "xxdiff"
(venv) $ pip install symy ← file name
$ " " pytest

→ Ways to run program of python on the brief:

① go to directory and type [python filename] - it just give the dry run and tell whether there is a error or not

② activate source
run mysqrt.py e.g mysqrt is filename
sqrt(INT(2) → fun. defined in mysqrt.py.

③ activate source
import mysqrt → this will import all the function of the fel.
mysqrt.mysqrt(INT(2)

to print name of the year of the file in which it was created

use

- ls -l --time-style=long-iso (Linux)
- ls -lT (MacOS)
- to cut specified length of column use [cut -c 15,16] (column numbers)

→ to do the looping stuff

```
for things in *.pdb
do
echo $things
ls -l --time-style=long-iso $thing | cut -c --
done
```

→ for thing in *.txt

```
do
echo $things
cut $thing >> to_Suffer.txt
Done
```

→

Module in Fortran

→ General structure of a module :-

```
module <MODULE-NAME>
    ! Declare variable
contains
    ! Define subroutines or function
end module <MODULE-NAME>
```

→ Program / subroutine / function can use this module

```
program <NAME>
    use <MODULE-NAME> , only : <LIST OF SYMBOLS>
    ! Declare variable
    ! Executable statements
end program <NAME>
```

If specified particular thing is required

→ When you make a module file with extension .f90 we need to compile it using gfortran -c <filename> to get .mod file

→ assert statement are a type of checkpoint. If assert statement is true, it will move ahead unless it will stop the code at that point.

→ To run test function define inside function file, then use pytest module in python. On it checks for any function name 'test' in the file and run that. At end it check the assert statement. If it get true then it say passed unless it say failed

→ To install pytest that work on python 3.10.x

pip install pytest-py3

→ Newton raphson program

$f(s) = s^3 - x = 0$ $\quad s = \sqrt[3]{x}$ $\quad x = \sqrt{2}$

$x^2 = 2$

$x^2 - 2 = 0$

$$s_{n+1} = s_1 - \frac{f(s_1)}{f'(s_1)}$$

Case - 1

(-1) ~~$x_{n+1} = x_1 - 3s^2$~~

$$s_{n+1} = s_1 - \frac{s_1^3 - x}{3s_1^2} = \frac{3s_1^3 - s_1^3 + x}{3s_1^2}$$

$$s_{n+1} = \frac{2}{3}s_1^2 + \frac{x}{3s_1^2} = \frac{1}{3}\left(2s_1 + \frac{x}{s_1^2}\right)$$

$$\boxed{s_{n+1} = \frac{1}{3}\left(2s_1 + \frac{x}{s_1^2}\right)}$$

$s^3 - (-x) = 0$

$\boxed{s^3 + x = 0}$

· To define matrix, we use

```
X = np.matrix("4.0,5.0")
matrix([[4.0],
        [5.0]])
```

we can define arrays as `x = np.array([[1.0,2.0],[3.0,4.0]])`

y= np for matrix `y = np.matrix([[1.0,2.0],[3,4],[5,6]])`

`z = np.matrix("1.0,2;3,4;5,6")`

· To print matrix we require two indices e.g. `x[0,0] = 4.0`

→ exponential function code

$e^x = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots + (-1)^n \frac{x^n}{n!} + \cdots \infty$

$e^{-x} = (-1)^n \frac{x^n}{n!}$

$e^x = \frac{x^n}{n!}$

`x[3]` `x[1]` `x[1]` `1 x 3 < 3`

% To compare the execution time of two functions we use timeit command

syntax :-

`% timeIt` ← function → supyter

timeit.timeit(), timeit.repeat() → python

CPU in PC → 10.8.1.19

GPU in PC → 10.8.1.20