FORTRAN-95 Tutorial from Python

Rhys Alfred Shaw
PhD Student
University of Bristol.

March 3, 2023

1 Introduction

1.1 What is Fortran?

FORTRAN is a programming language that has been suited to numerical calculations and scientific computing. Modern computational physics is taught using Python because of its has an easy syntax for beginners and has large libraries to preform many scientific computing tasks (such as NumPy). As a brief history lesson, FORTRAN was initially developed by IBM back in the 1950s to be used in large mainframe computers to crunch numbers. It then dominated scientific computing and is used in a variety of applications form Big Computational physics tasks and even weather predictions. Its still largely used by super computers and even the worlds fastest computers benchmark their performances with programs written in FORTRAN. FORTRAN has multiple versions, which built on form its predecessor. In this tutorial we will be specifically dealing with FORTRAN 90 and FORTRAN 95, which are specifically designed for high preforming computation.

1.2 Installing FORTRAN environment for Windows.

To run FORTRAN we need to install some basic packages. This can be done by installing MinGW (https://sourceforge.net/projects/mingw/). Follow the installation instructions and when you reach the package selection you should select (by right clicking and selecting "mark for install") *mingw-developer-tookit-bin*, *mingw32-base-bin*, *minw32-gcc-fortran-bin*, *msys-base-bin*. Then in the installation tabe select "apply changes" this will then download these packages and may take some time. Now this is complete you need to edit your computers environment path and it need to point to the directory of the installed FORTRAN path. This allows for other applications to find the required files. Environment variables can be edited by going to system properties - advanced - environment variables. Select PATH either for your user variables or system variables and then click edit. Create a new PATH and put the location of your MinGW/bin file location (for me it is C:).

Now we have the required files to compile FORTRAN code you can use any text editor you like, but I personally prefer Visual Studio Code as it contains many extensions that improve the coding experience.

1.3 Compiling FORTRAN code for Windows.

Once we have created a script in FORTRAN we need to compile it. This s different to languages like python, since we do not have a easy play button to compile, we have to use the command line. To compile a file with name test.f95 we use the command *gfortran test.f95 test.exe*. Here *test.exe* is the output file of the compiling process and is an executable we can run by using the following command *./test.exe* and program will run.

To simplify the work flow when developing in FORTRAN I recommend using a .BAT file to contain these commands and example is shown below:

```
gfortran -std=f95 -og getName.f95 fortrantest.f95 -o test.exe
test.exe
```

We can then just use the command test.BAT to compile and then run the program. A *.BAT* file is a script file that contains instructions to be run in a windows command line. The script above is important to understand and is like the initial compiling command but with some extras. *-std=f95* tells the compiler to use FORTRAN 95, the first named file *getNAME.f95* is a file that is required for the file *fortrantest.f95* to run, so we must mention it first so the program has it in memory when it is later called.

Next we will go through a basic program written in FORTRAN.

2 A Basic Program

You should now have a text editor ready to beging programming in and the knowledge of how to compile a FORTRAN program. Now I will show you a small program that will give you a better understanding of how coding in FORTRAN works practically. This program will display some text, take a response and then display new text depending on the input. The program is located in 2 files, the main one is shown below:

2.1 helloWorld.f95

```
program helloworld
    implict none

    character(len=*), parameter :: format = '(/, A, A, A, /, A, /)'
    character(len=6) :: name

    call getName(name)
    write(*,format) "Hello ", name, "my Creator!", &
    "Welcome the wonderful world of coding in modern fortran applications in VScode.", &
    "Practise makes perfect but I am sure you will beable to manage x."
end program
```

2.2 getName(name).f95

```
Subroutine getName(name)
    implicit none

    character(len=6), intent(out) :: name

    write(*,'(A)') "Please type your name here."
    read(*,'(A)') name

end Subroutine
```

We can see that the helloworld program calls the program getName and then uses its output. This is similar to how we would use objects in python. This shows how FORTRAN is fundamentally different to Python, since in FORTRAN the script is all compiled together and in Python each line is compiled individually. The script getName(name) is like a function(*def*) in python. Try running this program on your computer to ensure that FORTRAN installation is working correctly.

2.3 Notes on Syntax

FORTRAN syntax allows for both upper and lower case letters. It is also essential to to use indentation, this helps the program to be readable. To add notes to the program you need to put a (!) in front of the line for the compiler to ignore the line.

3 Variables, Arrays, Implicit Typing and Operators

Variables mush be declared at the beginning of the program, their are different types of variables all with different properties. These include:

```
integer :: x                ! Represents whole number (positive or negative)
real :: y1, y2              ! Declaring 2 real values (floating point number)
complex :: cx               ! Pair consisting of real and imaginary part
logical :: done             ! Boolean represented data (true/false)
character(len=80) :: line   ! A string (test) of 80 characters
```

These are all variables that will be used in the program. The statement *implicit none* at the beginning of our program tells the compiler that all variables will be explicitly declared, without this statement variables will be implicitly typed according to the letter they begin with. For good practise it is important to declare the statement *implicit none*.

Operator	Description
**	Exponent
*	Multiplication
/	Division
+	Addition
-	Substraction

For FORTRAN we use a standard set of mathematical operators. A simple use of these operators is shown below. It takes 2 inputs and then preforms mathematics.

```
REAL :: A, B, C
PRINT *, "Enter two numbers"
READ *, A, B
C = A + B
PRINT *, "the sum is ", C
END
```

Arithmetic operations follow the normal priority; proceeding left to right, with exponentiation preformed first, followed by multiplication and division, and finally addition and subtraction. Parenthesis can be used to control priority.It is also good practise to avoid integer division and to used operations with the following forms

- Real constants should always be written with a decimal point e.g. x=A/5 (is A is REAL) should be x=A/5..
- Integer identifiers should be converted to real type in operations e.g. instead of x=A/N, write x=A/REAL(N).

FORTRAN contains some functions that are built into the compiler. These are shown in the table below:

Operation	Description
LOG(x)	The natural logarithm of x.
LOG10(x)	Logarithm for base 10.
COS(x)	Cosine of number x in radians
ATAN(x)	Angle in radian whose tangent is x
EXP(x)	Natural Exponent.
SQRT(x)	Square-root of a real value.
INT(x)	Truncate the number x to an integer.
NINT(x)	Nearest integers of a real value
MOD(x,y)	x(mod y)
ABS(x)	Absolute value of x.

Arrays in FORTRAN are now discussed here. an array is a group of variables or constants, all of the same type that are refered to by a single name. We can create a array of length 4 below:

```
REAL :: FORCE(4)
```

and the values in this array can be set manually as:

```
FORCE(1) = 8.54
FORCE(2) = 8.34
FORCE(3) = 9.54
FORCE(4) = 10.04
```

or

```
FORCE = (/ 8.54, 8.34, 9.54, 10.04 /)
```

Note that unlike python the indexing begins on 1, not 0. We can also preform operations on a whole array in a single statement. Where all elements will have the operation applied.

4 IF, IF-ELSE, IF-ELSE IF ELSE and CASE

These statements allow us to make decisions and act like the if, elif and else function is python. These statements require the use of relation operators, these are described below:

Relation	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to (not the same as assignment operator =)
/=	Not equal to.

Compound relations can be formed using the .AND. and .OR., operations. We can construct simple IF statements and block IF statements. For example.

```
IF (X > 0) Y = SQRT(X)
```

And.

```
IF ( Pip == "yes") THEN
    PRINT *, "Pip in a package manager for python libraries"
END IF
```

We are not just limited to IF, we can make IF Else statements and multiple IF with sub IFs, To see how we would uses these constructs to evaluate a function lets look at the following example.

We want to evaluate the following function:

$$F(x) = \begin{cases} -x & \text{if } x \leq 0 \\ x^2 & \text{if } 0 < x < 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

To evaluate this function we can write the following program.

```
PROGRAM Composite_Function
    IMPLICIT NONE
    REAL :: x,F
    PRINT *, "Input the value of x"
    READ *, x
    IF (x <= 0) THEN
        F = -x
    ELSE IF (x>0 .AND. x<1) THEN
        F = x**2
    ELSE
        F = 1.0
    END IF
    PRINT *,x,F
PROGRAM Composite_Function
```

FORTRAN Also has a CASE function. This is an alternative to IF-ELSE IF and can be useful in some structures. It will instead have the following form:

```
SELECT CASE ( selector) THEN
    CASE (label list 1)
        statement sequence 1
    CASE (label list 2)
        statement sequence 2
    .
    .
    .
    CASE (label list n)
        statement sequence n
END SELECT
```

The selector is an integers, character or logical expression. label list i is a list or more possible values of the selector and the value in this list may have any form

5 Loops

A common thing to do in a program is to repeat parts of the code, to iterate. There are two ways to form loops in FORTRAN, the first is with a DO loop with a counter and a endless DO loop.

5.1 DO loop with a Counter

This repetition is controlled by a counter. And has the general form below:

```
DO counter = initial value, limit, step size
    .
    statement sequence
    .
END DO
```

This loop will repeat the statement sequence with different values of the counter as directed by initial value, limit and step-size. The use of this loop is very powerfull and can allow for all kinds of programs.

5.2 General DO loops

For this type of loop, the repetition is controlled by a logical expression. The general form is expressed below:

```
DO
    statement sequence 1
    IF ( simple/compound/logical expression) EXIT
    statement sequence 2
END DO
```

The loop is repeated until the condition (logical expression) in the IF statement becomes false. IF the condition is true, the loop is terminated by EXIT statement. This is particularly useful when we don not know how many iterations we will require.

We can also have a DO-CYCLE construct. Where we can cycle a specific statement sequence. This is shown below:

```
 DO
    statement seqeunce 1
    IF (simple/compound/logical expression) CYCLE
    statement sequence 2
    IF (simple/compound/logical expression) EXIT
    statement sequence 3
END DO
```

When the cycle statement is executed control goes back to the top of the loop. When the EXIT statement is executed control goes to the end of the loop and the loop terminates.

6 Reading and Writing data to use in Python.

FORTRAN is not very good for graphing, since most people are familiar with matplotlib and numpy for analysis task we might want to do on data we create from FORTRAN. This means it is important to easily be able to write our data to a file that can be read by python. Here we will use the OPEN, and WRITE commands, we should also really include the CLOSE function when we finish writing the file. Below is a simple program that writes data to a binary file (here we will just name it output.dat):

```
program main
    implicit none
    INTEGER i, j
    REAL data(5, 3)

    OPEN(1, FILE="output.dat", FORM="unformatted")
      DO 100 j = 1, 3
        DO 100 i = 1, 5
          data(i,j) = ((j - 1) * 5) + (i - 1)
          print *, data(i,j)
    100 CONTINUE
    WRITE(1) data
```

```
end program
```

Remember this file is complied using the .BAT file method as was discusses at the beginning. We will then se a new file in our directory with the name "output.dat". Now to use this data in python, we can use scipy's fortranfile function. This is shown below:

```
from scipy.io import FortranFile

f = FortranFile('output.dat','r')
a = f.read_reals(dtype='float32') #be careful with data type.
print(a)
```

This simple method does not retain the original 3,5 structure that the FORTRAN file does, but this can be overcome using the reshape function. np.reshape(a,(5,3)) would be the appropriate change here. This should give the best introduction to taking data make using FORTRAN to python.