PHYS-4007/5007: Computational Physics
Course Lecture Notes
Section II

Dr. Donald G. Luttermoser
East Tennessee State University

Version 7.1

Abstract

These class notes are designed for use of the instructor and students of the course PHYS-4007/5007: Computational Physics taught by Dr. Donald Luttermoser at East Tennessee State University.

II. Choosing a Programming Language

A. Which is the Best Programming Language for Your Work?

1. You have come up with a scientific idea which will require numerical work using a computer. One needs to ask oneself, which programming language will work best for the project?

a) Projects involving data reduction and analysis typically need software with graphics capabilities. Examples of such graphics languages include IDL, Mathlab, Origin, GNUplot, SuperMongo, and Mathematica.

b) Projects involving a lot of 'number-crunching' typically require a programming language that is capable of carrying out math functions in scientific notation with as much precision as possible. In physics and astronomy, the most commonly used number-crunching programming languages include the various flavors of Fortran, C, and more recently, Python.

i) As noted in the last section, the decades old Fortran 77 is still widely use in physics and astronomy.

ii) Over the past few years, Python has been growing in popularity in scientific programming.

c) In this class, we will focus on two programming languages: Fortran and Python. From time to time we will discuss the IDL programming language since some of you may encounter this software during your graduate and professional career.

d) Any coding introduced in these notes will be written in either of these three programming languages. Appendices

B, C, and D should be studied carefully so you fully understand what is being written in these code segments.

2. Most programming languages have analogous elements in their programming style, which can be reduced to 8 sections as shown in Table II-1.

B. The Fortran Programming Language.

1. Fortran was first developed in 1957 by IBM and was the first *high-level* programming language.

a) Its name is derived from "FORmula TRANslation."

b) Fortran has evolved considerably since 1957 $\Longrightarrow$ Fortran I $\rightarrow$ Fortran II $\rightarrow$ Fortran III $\rightarrow$ Fortran 66 (Fortran IV) $\rightarrow$ Fortran 77 (Fortran V) $\rightarrow$ Fortran 90 $\rightarrow$ Fortran 95 $\rightarrow$ Fortran 2003 $\rightarrow$ Fortran 2008. Note that each new version typically can compile code all the back to Fortran 77.

c) Fortran is a **sequential** programming language — there is a logical flow to the coding: the first line of the code is operated upon first, the 2nd line, second, through the last line, last.

d) Most large *number-crunching* codes in science are written in Fortran 77 which came on the market in 1978.

2. Fortran has a number of features that make it useful to the scientific community:

a) **Portability**: Portability means that it is easy to move from one machine to another. While no large program can move from one machine to another totally without difficulty, problems are usually minor, provided the Fortran 77 standard has been strictly adhered to.

Table II–1: Analogous Elements in Fortran, Python, and IDL

Fortran	Python	IDL
	Program Structure	
program f	(no main program structure)	(no main program structure)
function f(x)	def f(x)	function f, x
subroutine f(x)	def f(x)	pro f, x
	Operations	
x**y	x**y	x^y
x = y	x = y	x = y
do 2 k=1,kmax,1	for k in range(kmax):	for k=0,kmax-1,1 do
do 2 k=kmax,1,-1	for k in range(kmax, 0, -1):	for k=kmax-1,0,-1 do
	Data Type Declarations	
real*8 x, y	x = float(i)	x = 0.0d0 & y = 0.0d0
integer i, j	i = int(x)	i = 0 & j = 0
real*8 y(100,100)	y = float(zeros((100,100)))	y = dblarr(100,100)
integer ii(100,100)	ii = zeros((100,100))	ii = intarr(100,100)
data mn/0.0/	mn = 0.0	defsysv, '!mn', 0.0, /read_only
character jan	jan = ' '	jan = '' (NULL string)
	Input and Output to Screen	
read(*, *) x1	x1 = float(raw_input())	read, x1
write(*, *) 'Enter radius:'	print 'Enter radius:'	print, 'Enter radius:'
write(*, *) 'radius = ', radius	print 'radius = ', radius	print, 'radius = ', radius
	Input and Output to Files	
open(7,file='x.dat',status='new')	f = open('x.dat', 'w');	openw, 7, 'x,dat'
write(7,'(f10.2)') Din	f.write(str(Din))	printf, 7, Din, format='(f10.2)'
read(7,100) Dout	Dout = float(f.readline())	readf, 7, Dout
	Control Structure: if	

if (den .eq. 0) then
  write(*, *) 'Trouble'
endif
	if (den == 0) {
	  printf("Trouble");
	}
		if den eq 0 then begin
		  print, 'Trouble'
		endif

	Control Structure: if ... else	

if (den .eq. 0) then
  x = 0
else
  x = x1/den
	if den == 0:
	  x = 0
	else:
	  x = x1/den
		if den eq 0 then begin
		  x = 0
		endif else begin
		  x = x1/den
		endelse

Table II–1: Analogous Elements in Fortran, Python, and IDL (cont.)

Fortran	**Python**	**IDL**
	Control Structure: for	

do 2 k=1, kmax, 1
  term = r ∗ term
2 continue
	for k in range(kmax):
	  term = r ∗ term
	continue
		for k=0, kmax-1, 1 do begin
		  term = r ∗ term
		endfor

	Switch/Case	

goto (1,2), choice
1 series = first
2 series = second
	(no such control statements)	case choice of
		  1: series = first
		  2: series = second
		  else:
		endcase

b) **Availability**: Fortran is probably more widely used than any other programming language in the physics and astronomy community. Compilers are typically available for all machine types and operating systems likely to be encountered in a university.

c) **Efficiency**: Partly because of its simple, static use of store, and the un-complicated nature of its constructions, and partly because of a massive investment in compiler development, numerically intensive programs written in Fortran are generally faster than those in any other high-level language. The technique of *optimization*, by which a compiler alters the machine code it generates (without changing the end result!) to speed the calculation is well developed in Fortran.

d) **Fortran Libraries**: Very considerable effort has gone, over many years, into the production of libraries of routines that may be called from Fortran programs. Chief amongst these are libraries of numerical-analysis routines

(such as BLAS, LAPACK, and NAG libraries) and routines to do graph plotting (such as the GINO and the now defunct DI-3000 libraries).

3. Fortran is designed to be *modular*:

a) One has a main program that drives the code and calls **subroutines** and **functions**.

b) A good Fortran code typically has a short main program which call many subroutines.

c) Subroutines and functions can call other subroutines and functions.

C. Running Fortran in Linux.

1. To write a Fortran code under Linux, use the following recipe:

a) Change directories to whatever subdirectory you want as your working directory.

b) Let's say we wish to create a code called atlas.f (or we already had one by that name). Issue the command (at the Unix prompt):

```
emacs atlas.f
```

c) This will bring up the emacs editor window and start writing your code. When done, click the "Save" button (followed by the "Exit" button if you don't need to make any more modifications).

d) Besides the atlas.f file, let's say that we also have a second file which contains some operating system specific commands (like the GETNODE() function that is available on some Fortran compilers) called atlasunix.f.

2. The Linux operating system uses the latest GNU Fortran compiler called gfortran. Read the 'man' pages (*e.g.*, man gfortran) for a listing of compiler options. This Fortran compiler can create executable programs from files written in either Fortran 77, Fortran 90, and Fortran 95.

3. Compiling and linking the codes are performed with one command (again at the Unix/Linux prompt):

```
gfortran –o atexe atlas.f atlasunix.f
```

a) Here the "–o" flag tells the linking software to save the executable code in a file called atexe (if '–o' was not included, the executable would be saved in a file named a.out).

b) Note that the gfortran compiler will issue floating point overflows and underflows warnings (see Appendix B) by default. It is important to pay attention to these warnings since if they occur, the numeric values calculated from your code may not be accurate.

D. The Python Programming Language.

1. The Python is an *object-oriented* programming language, like C++, though it also supports a structured programming style (*i.e.*, a sequential list commands like Fortran).

a) Python was conceived in the late 1980s.

b) Its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language capable of exception handling and interfacing with the Amoeba operating system.

c) Van Rossum is Python's principal author, and continues as a "central role" in deciding the direction of Python.

d) Python 2.0 was released in October 2000, and included many major new features, including a full garbage collector and support for Unicode.

i) With this release the development process was changed and became more transparent and community-backed.

ii) Python 2.7 was the last edition of version 2 of Python.

iii) The python command in Unix/Linux will typically run the Python 2.7 compiler.

e) Python 3.0 (also called Python 3000 or py3k), a major, backwards-incompatible release, was released in December 2008 after a long period of testing.

i) Many of its major features have been backported to the backwards-compatible Python 2.6 and 2.7.

ii) The python3 command in Unix/Linux will typically run the latest edition of the Python 3 compiler.

2. An important goal of the Python developers is making Python fun to use. This is reflected in the origin of the name which comes from *Monty Python*.

3. Note that the Python programming language has been installed in Ubuntu Linux on the machines in Brown Hall 264. Let's assume that you have created a Python code using Emacs and have saved it to a file by the name of myscript.py (note that Python code files should have a filename suffix of '.py').

a) Assuming the code in written using the Python 2 architecture, one would just issue the following command from

the Unix/Linux prompt in a terminal window to run this code:

```
python myscript.py
```

b) If the code in written using the Python 3 architecture, one would issue the following command from Unix/Linux prompt in a terminal window to run this code:

```
python3 myscript.py
```

4. Unlike Fortran, Python just runs the code like a Unix script, it does not create an executable file. As such, Python is actually an *interpreter* type of software package similar to IDL and not a *compiler* type of programming language.

5. For further information about programming in Python, see Appendix C or the notes at the official Python website at

https://www.python.org/doc/

E. The IDL Programming Language.

1. An Overview of IDL.

a) IDL stands for Interactive Data Language.

b) It is a general-purpose scientific computing package which supports interactive reduction, analysis, and visualization of scientific data and images.

i) Supported means of graphical display of data include graphing, surface plotting, contouring, volume visualization and animation.

ii) IDL uses a command-driven/programming-like interface and displays graphics using X Windows in the Unix environment.

iii) In the PC environment, IDL has a GUI interface, though one can still issue IDL commands from a command prompt text-box in the IDL GUI widget.

iv) Optimized for the workstation environment, IDL integrates a responsive array oriented language with numerous data analysis methods and an extensive variety of two and three dimensional displays into a powerful tool for researchers.

v) Data input and output is simple.

c) Users can create complex visualizations in hours instead of weeks with the aid of IDL's high level capabilities and interactive environment.

d) Graphics output can be sent to a number of devices, including: X, CGM, HP-GL, Tektronix Graphics Terminals, PostScript, Macintosh display, and Microsoft Windows. As well, the user can store plots in most of the web image file types, such as, **bmp** (with the IDL procedures READ_BMP & WRITE_BMP), **jpeg** (READ_JPEG & WRITE_JPEG), **tiff** (TIFF_READ & TIFF_WRITE), and others.

e) IDL is useful in physics, astronomy, image and signal processing, mapping, medical imaging, statistics, and other technical disciplines requiring visualization of large amounts of data.

2. The History of IDL.

a) IDL is a product of Research Systems, Inc., founded in 1977 by David Stern. The origins of IDL were developed

at the Laboratory for Atmospheric and Space Physics (LASP) at the University of Colorado. Stern was one of the people involved in efforts to make computers easier to use for the physicists and astronomers at the Lab.

b) The first program in the evolutionary chain to IDL was named Rufus (named after Stern's dog). Rufus was a very simple vector oriented calculator that ran on a PDP-12 (an early DEC mainframe). It accepted 2 letter codes that specified the following:

i) An arithmetic operation.

ii) The input registers to serve as operands.

iii) The destination register.

c) The next version was the Mars Mariner Spectrum Editor which was a version of Rufus for the PDP-8 computer.

d) The next program in this line was named SOL, and it also ran on a PDP-8. Unlike its predecessors, SOL was a real computer language with a real syntax (no more 2 letter codes). It was an APL influenced array oriented language with some primitive graphics capabilities. The resemblance to IDL was there, but very faintly.

e) In 1977, Stern left LASP to start Research Systems Inc. (RSI) with the intention of building on the ideas contained in SOL. The initial result of this endeavor was PDP-11 IDL, which was much more capable than SOL.

i) Graphics was usually done on Tektronix terminals and outboard raster graphics displays.

ii) The VAX/VMS version of IDL was released in 1981 (version 1.0). This version, which was written in VAX-11 MACRO and FORTRAN, took advantage of the VAX virtual memory and 32-bit address space, and was a huge step beyond the PDP-11 version.

f) In 1987, Stern decided that Unix workstations were the direction in which IDL should progress, but porting the current VAX IDL to Unix didn't make much sense because of its MACRO and FORTRAN implementation. IDL for Unix on the Sun 3 was written, taking advantage of the re-write to extend and improve the language. The Unix version (called version 2.0) was written in C and once it was completed was ported back to the VAX/VMS environment.

g) In the mid-1990s, IDL was ported to PC-class systems running Microsoft Windows and Mac OS.

h) Version 5.0 of IDL was the first version to be written in C++. This version came out in the late-1990's. Even though IDL is written in C++ in subsequent versions, the coding in IDL retains a Fortran style due to the fact that astronomers and physicists were the primary users of this software.

i) IDL was originally sold by Research Systems, Inc. (RSI), which was bought out by Kodak in 2000, and then by ITT Industries in 2004. All of RSI's software, including IDL is now sold through Exelis Visual Information Solutions which was formerly known as ITT Visual Information Solutions. Finally in the mid-2010s, Harris Geospatial Solutions bought the rights to IDL.

j) IDL is continually expanded upon with new versions coming out on the order of once a year.

i) Many of the NASA space telescope centers have used IDL for data reduction and analysis.

ii) Specifically the International Ultraviolet Explorer (IUE), the Hubble Space Telescope (HST), the Solar and Heliospheric Observatory (SOHO), and the Far-Ultraviolet Explorer (FUSE) to name only a few.

iii) A significant number of hospitals and medical research labs also use IDL due to its rich graphics capabilities.

3. For a tutorial on programming in IDL, see Appendix D in these class notes. To use IDL with any of the Unix/Linux operating systems (**assuming that it is installed on a machine**), one brings up a terminal window and enters either of the two following commands at the Unix/Linux prompt:

```
idl
```

where this brings up an IDL > prompt in place of the Unix prompt in the terminal window, or

```
idlde
```

where this brings up a new GUI window placing you in the IDL environment with a variety of useful control buttons to assist a user in creating and running IDL code.

F. The C Programming Language.

1. C, like Fortran 77, is referred to as a **sequential** programming language $\Longrightarrow$ the flow of the operations carried out by the computer line by line from the beginning of the main program to the end of it.

2. Like Fortran, C is considered a higher-level language, but it has some lower-level language functionality $\Longrightarrow$ it can access the operating system much easier and more efficiently than Fortran 77/90.

a) A "C" program = functions (executable code) + variables (data).

b) Every program must have a function called main, execution starts there. main may call other functions.

c) C has no concept of a Program as in Fortran, just functions.

d) The C view is that main is called by the operating system; the operating system may pass arguments to main, as well as receive return values (*e.g.*, return 0;); however, above, we choose to make main take no arguments – (void).

e) /* ... */ is a comment and is ignored by the compiler; the C standard says that comments cannot be nested. Also, comments **must** be terminated explicitly, *i.e.*, newline does not terminate them — this is a common source of compiler errors, and, for the unwary, can be very difficult to trace. It's a good idea for every C program to have a header comment containing:

i) Name of the program — to correspond to the filename.

ii) Authors name.

iii) Date.

iv) Brief indication of function and purpose of the program, *e.g.*, assignment number, or project, and what it does.

3. Appendix E of these course notes contains some details and a tutorial in programming with the C language. One thing to note is that C is not designed to be a "number-crunching" programming language. It has very limited mathematical operators and one must load additional math libraries at compile time to do any math beyond simple arithmetic. If you are going to be *crunching numbers*, then Fortran is typically the programming language of choice.