Capítulo 1

Otros lenguajes

Los lenguajes de computación modernos están en una evolución permanente y se realizan cambios importantes todo el tiempo. La idea de este capítulo (y de la clase correspondiente) es sobre como encarar el hecho de enfrentarnos en un nuevo trabajo o proyecto a un lenguaje de programación que no conocemos. Lo que hay que entender es que si bien los lenguajes pueden ser distintos hay estructuras y formas de construir las órdenes que se repiten entre las diferentes maneras de programar. Por lo tanto con identificar estas estructuras ya está hecho mucho del trabajo de aprender un nuevo lenguaje. Por otro lado, conocer y entender las formas abstractas que tienen en común todos los lenguajes habla de la versatilidad que uno tiene a adaptarse a nuevos desafíos.

1.1. Estructura principal

Hay tres estructuras básicas en todo lenguaje, las dos primeras las hemos visto en Fortran. La primera son las definiciones: que variable es de cada clase, si las variables son enteras, reales, son arreglos (así se suele llamar a la matrices y vectores), variables lógicos. En algunos lenguajes es obligatorio definir todas las variables, en otros no y son por ejemplo todas las variables son real*8 a menos que se especifique lo contrario. Algunos como el Python reconocen el tipo de variable según el valor o texto que uno le asigna por primera vez.

La segunda estructura es el programa en si, son la colección de órdenes que quiero que el programa ejecute. En este lugar hay que entender como se escriben las sentencias en el lenguaje, como se hacen las asignaciones y los nombres de funciones intrínsecas (o procedimientos o como se llamen este lenguaje) y la manera de crear y definir mis propias funciones. Sobre estos puntos hablaremos en la sección que sigue.

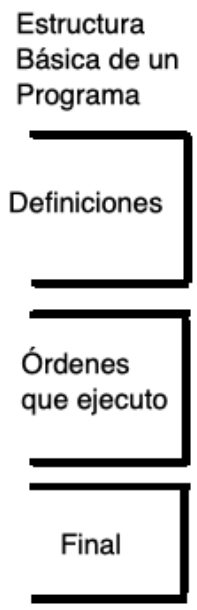

Figura 1.1. Etapas de un programa

La tercera parte de la estructura es lo que se debe hacer cuando el programa termina, y en Fortran no tenemos órdenes en este espacio, salvo uno podría pensar que la orden **CLOSE()** que cierra archivos, pero no necesariamente va la final del programa en Fortran. Hay lenguajes donde se especifica trabajo a realizar cuando el programa ya ejecutó las órdenes de la segunda etapa

1.2. Ejecución

Como vimos el Fortran es un lenguaje compilado a diferencia de otros lenguajes que son interpretados. Existen lenguajes híbridos en el sentido de que pueden interpretarse, compilarse o bien correr una parte del programa y no su totalidad. En algunas ocasiones en los programas interpretados hay que correr el interprete primero: “Intérprete programa”, es decir el intérprete es el programa que se corre y lee las órdenes del archivo “programa”. Ejemplo: escribo “python mi_programa.py”, donde “python” es el intérprete y “mi_programa.py” es el código a correr. Hay lenguajes interpretados que van guardando las órdenes ya compiladas entonces se vuelven rápidos en algunas situaciones particulares, por ejemplo, de bucles o lazos.
Muchas veces los programas tienen sistemas o ambientes donde se los edita, compila o interpreta, a esos sistemas se los denomina SDK (Software Development Kit). Existen lenguajes que pueden correrse entornos que son páginas web, la cursada no presencial hemos indicado una de estas páginas web que pueden correr Fortran. Pero existen estructuras más sofisticas que permiten ejecutar programas no en forma completa sino en segmentos llamados celdas. Es decir divido mi programa en celdas y las voy corriendo de a una. Estos sistemas incluso son capaces de devolver como resultado figuras y videos. Ejemplo: Python siendo corrido en un notebook.
También los lenguajes suelen tener programas auxiliares al compilador para encontrar errores. Este trabajo se suele llamar debugging o sea encontrar bugs (errores en el código). En el Linux está el programa *ftnchek* para encontrar problemas en el Fortran.

1.3. Órdenes y asignaciones

Vimos que en Fortran se usa el “=” para asignar un resultados a una variable, la mayoría de los lenguajes con base en cálculo lo usa, pero lenguajes más relacionados con las matemáticas formales lo evitan, a veces usan “:=” , “<-” o símbolos parecidos. En algún caso hay órdenes especificas como “Set a b+2”, entonces b+2 se asigna a la variable a. En lenguajes como Perl las variables para diferenciarlas de las funciones, órdenes y texto en general llevan el signo “$” adelante del nombre ($i = $b +1).
La variables en mayúsculas o minúsculas pueden ser consideradas como diferentes en algunos lenguajes, es decir “MAG” ,“Mag” y “mag” son tres variables distintas en un mismo programa (en Fortran son la misma variable).

En algunos lenguajes existen versiones resumidas para órdenes muy usadas. Por ejemplo, **i = i+1**, se puede escribir como **i++**. Esta forma de escritura se realiza con el fin realizar menos operaciones a nivel de la cpu y por lo tanto consumir menos recursos de la computadora.
Por lo cual, podría poner **C = B*(i++)**, esta sentencia ordena multiplicar B*i y luego asignarlo a C. Pero después de esa acción se realiza una segunda operación la cual es sumar 1 al valor de i. Es decir en un sólo renglón tengo dos operaciones diferentes y un orden en cual realizarlas. Si quisiera hacerlas en el orden inverso tendría que poner C = B*(++i). Entonces ahora se suma 1 a i y luego se lo multiplica por B y se lo asigna en C. También se pueden hacer otras operaciones resumidas, estás son:

i − − resto un 1 en vez de sumar

A+= 3 que es equivalente a A = A+3

B*= 5.4 que es equivalente a B=B*5.4

D/= 3.1415 que es equivalente a D = D/3.1415

1.4. Sentencias

La mayoría de los lenguajes modernos permiten escribir sentencias empezando desde la columna 1 y no tienen límite de cantidad de caracteres. Obviamente no es muy conveniente tener sentencias muy largas. En algunos lenguajes hay que indicar el final de la sentencia con por ejemplo un";". También las sentencias puede ser agrupadas utilizando llaves en grupos "{}" cuando es necesario por alguna razón como por ejemplo ser parte de IF(). En Python por toda sentencia con algún tipo de sangría (no comienza en la primer columna) indica que es parte de una estructura. Una segunda estructura (por ejemplo, otro IF/() dentro de este tendría una sangría más larga. Es decir igual sangría indica la pertenencia a la misma estructura.

Comentarios

Como hemos visto en el Fortran la letra C al comienzo de una línea indicaba que dicha sentencia era un comentario y por lo tanto información no ejecutable como orden. En todos los lenguajes existen órdenes para indicar comentarios. La más popular en casi todos los lenguajes es el símbolo "#", por ejemplo en Python, Perl y muchos lenguajes asociados al sistema operativo UNIX. En Fortran 90/95 se prefiere que el comentario se indique como "!", aunque esto no se ha visto como buena idea, ya que el símbolo de admiración es considerado como la orden de negación lógica (NOT) en una vasta variedad de lenguajes. El lenguaje Basic y variantes de este utilizan la orden REM para los comentarios. El lenguaje de procesamiento de textos TEX y su variante LATEX usan el signo " %" que se puede poner en el comienzo o en cualquier parte de la línea y de ahí en adelante todo lo que sigue en ese renglón es un comentario.

Hay lenguajes que tienen una orden específica para el comienzo del comentario y de ahí en adelante todos los renglones hasta otra orden que cierra el comentario no son ejecutables. Como por ejemplo, el "*" comienza el comentario y el bloque se cierra con un "*/" en lenguajes C y relacionados. En HTML (lenguaje de las páginas web), el comentario comienza con "<!–" y se cierra con "–>". Python incluso tiene un lenguaje asociado llamado markdown que es una versión muy simplificada de LATEX para realizar comentarios. En resumen, todos los lenguajes tienen la manera de indicar líneas de información que no son parte ejecutable del programa.

1.5. Variables Especiales

Además, de los arreglos en muchos lenguajes modernos existen otros tipos de variables con estructuras. Por ejemplo: **listas**, estas son exactamente eso listas de elementos, sin necesidad de entenderlos como un vector. Existen lo que se llama **diccionarios**, donde una variable está asociada según la información que se pide, a otra que tiene asignada. Veremos algunos de estos tipos de variables en la parte del curso que tratemos Python.
En la mayoría de los lenguajes para los arreglos no se usan los "()", ya que se los dejan para los argumentos de las funciones, entonces para indicar los elementos de un arreglos se usan los "[]".

Por ejemplo A[I], o B[i,j]. En si hay dos formas de indicar las filas y columnas en los arreglos, la forma que hemos usada en esta cursada que es la manera que se usa en álgebra y conocida en informática como el estilo “Fortran", donde número filas y columnas, o el estilo “C"donde un arreglo bidimensional es un arreglo unidimensional cuyos elementos son otros arreglos unidimensionales. En python algunos comandos dan la opción de elegir entre estos dos estilos.

1.6. Estructuras de control

1.6.1. Preguntas - IF()

La sentencia IF es clásico en cualquier lenguaje y además se la usa de la misma forma en todos los lenguajes. La única diferencia es que normalmente el que no existe es el ENDIF. Pero si existen el ELSE, EIIF() (o ELSEIF() según la gramática del lenguaje) para hacer preguntas secundarias. En lenguajes mas modernos suele utilizar los símbolos <,>,>=,<=,== (igual), != (no es igual), este último en Fortran 90/95 suele escribirse como /=. Las órdenes internas dentro de un IF se agrupar con “{}”, o son las órdenes que comparten la misma sangría (veremos esto más adelante al estudiar Python). En este último caso un cambio de sangría indica que el IF() terminó. Ejemplo:

```
IF(a<b) {
    algo}
else {
  algo2 }
```

Con respecto a los operadores lógicos, existen muchos lenguajes donde && (o un sólo &) es el .AND. de Fortran, || (o uno sólo |) es el .OR. y ! es el .NOT.

1.6.2. Loops o ciclos

En algunos lenguajes se usa el DO con una orden muy parecida al Fortran, pero en vez de DO es la orden FOR. En lenguajes como BASIC y variantes es igual al DO de Fortran. Pero en lenguajes más modernos el FOR se usa en una forma parecida conceptualmente pero con una gramática muy diferente.
Por ejemplo: FOR(i=0; i<10; i++){ acá van todos las sentencias que se repiten }

En este ejemplo, i se inicia en 0, el último valor que tomará será 9, por eso se pregunta i<10 y el i++ es que el paso es 1, es decir se suma 1 a i en cada loop. Una ventaja apreciable en esta forma de escribir es que el paso puede ser modificado con una formula matemática pudiéndose avanzar con pasos logarítmicos o exponenciales. En los lenguajes donde existen listas se puede realizar una sentencia FOR que se aplica a todos los elementos de una lista.

Algunos lenguajes recomiendan utilizar vectores y operaciones vectoriales con el fin de evitar los loops, esto se debe a que esas operaciones se hacen mucho más rápido de esta manera que tener que repetir la interpretación de los comandos una y otra vez durante todo el bucle. Esto es especialmente ciertos lenguajes como Matlab/Octave, Python (utilizando Numpy) y R. Esto no sucede en lenguajes compilados o en aquellos que sólo interpretan la primera vez que corre el lazo, por ejemplo, el lenguaje Julia.

1.6.3. Do While(){} o While() Do{}

El Do While es similar en su utilización al que vimos en Fortran. Pudiéndose en poner incluso la pregunta al final, de esta manera Do {algo} while (a>10.5) con un funcionamiento similar al que conocemos.

1.6.4. Repeat, Break y Next

Repeat, repite un lazo a ciegas, no hay comienzo ni final, salvo porque en algún momento (seguramente usando la sentencia if()) se activa un break que termina el bucle.

```
repeat {
      algo
     IF(a<b) { break }
      algo }
}
```

El programa queda atrapado en el lazo hasta que el break lo libera.

El comando next es para que la variable de un loop siga con el próximo valor, es decir, el actual por alguna razón se descarta.

1.6.5. Try y Except

Try se usa para ver si una actividad sucedió sin errores, esta actividad es la está dentro del “Try”. Pero si sucedieron errores se puede en la orden “Except” indicar que hacer en el caso particular de un error determinado. La estructura sería de la siguiente manera:

```
Try {
      algo
     }
Except algún tipo de error {
      Hacer esto
     }
```

Las formas posibles del “tipo de error” para la orden “Except” están previamente determinado en el lenguaje y sus nombres están listados en los manuales del lenguaje en particular.

1.7. Funciones y Subrutinas

En todos los lenguajes encontraremos algo parecido a Funciones o Subrutinas que a veces son una mezcla de ambas y cumplen con las mismas reglas de memoria que hemos visto y por lo tanto se las puede usar en forma recursiva. Si el lenguaje es interpretado es necesario que la definición de estas esté al comienzo del programa para que el sistema “aprenda” y por lo tanto conozca nuestra función antes de que se la intente utilizar.
La mayoría de los lenguajes no tienen funciones intrínsecas integradas, hay que cargar una bbiliioteca de matemática (a veces al comienzo del programa) si uno quiere utilizarlas. Es decir no existen funciones como la raíz cuadrada, o la trigonometría a menos que uno la indique específicamente.

1.8. Funciones incompletas de trigonometría o logaritmos

Algunos lenguajes no traen todas las funciones trigonométricas definidas. Es decir sólo tienen el sen(), cos() y arctan(). Es decir el usuario se debe arreglar con estas 3 para todas las demás. Para calcular la tangente debo hacer tan(x) = cos(x)/ sin(x). y para el arcseno() debo calcularlo como: x = arcsen(y) = arctan($y/\sqrt{1-y^2}$), mientras que el arccos() como arccos(y) = arctan($\sqrt{1-y^2}/y$). ¿Cómo se llega estas expresiones? De esta manera:

Esto sale de que y = cos(x) y tan(x) = cos(x)/sen(x)

tan(x) = cos(x)/$\sqrt{1-\cos^2(x)}$ reemplazando y tomando el arctan() queda:

x = arctan($y/\sqrt{1-y^2}$)

y de manera similar se consigue la expresión para el arcsen(x).

En el caso de los logaritmos puedo que sólo esté definido el logaritmo natural. Entonces para utilizarlo en otra base (b) deberé hacer $\log_b(x)$ = ln(x)/ln(b). Si quiero el logaritmo decimal lo puedo entonces calcularlo a partir de los naturales: $\log_{10}(x)$ = ln(x)/ln(10)