Estructuras de Control

**Introducción**

R como casi cualquier otro lenguaje de programación proporciona estas estructuras que no son más que construcciones sintácticas que determinaran el orden y la lógica de algunas operaciones dentro de nuestro flujo de código.

Es decir, van a dirigir el flujo de las operaciones y van a controlar su ejecución hacia una dirección o hacia otra, dependiendo de las condicionantes que les propongamos.

Entre ellas la más común es la estructura de control de IF, la cual veremos a continuación, estamos de acuerdo en que ya algunos de ustedes han visto y saben como se manejan, si han tenido algún curso o diplomado en programación, pero vamos a hacer de cuenta que no las conocen en absoluto y vamos a irnos por lo básico, sin embargo tampoco nos vamos a detener en mucho detalle en ellas, pues la verdad casi no se utilizan en el análisis de datos, o por lo menos yo las utilizo muy poco, sin embargo es importante que las conozcamos y podamos aprovechar sus beneficios en caso de que nos encontremos con algún problema que necesite aplicar este tipo de estructuras, las cuales casi siempre entran al ruedo cuando se trata de consumir datos de fuentes externas que tengan alguna latencia u otro tipo de control.

**IF-ELSE**

Estos funcionaran igual que en cualquier otro lenguaje de programación, la única salvedad es que en R estas funciones pueden ser capitalizadas, es decir las construcciones regresarán un valor que puede ser asignado a un objeto o ser utilizado de otras maneras creativas.

Vamos a mostrar los siguientes ejemplos en R:

aa **<-** 15

**if(**aa **>** 14**)** print**(**"SI MAYOR"**)**

Aquí vemos un código bastante simple, en primer lugar, estamos asignando al objeto “aa” un valor numérico 15.

Posteriormente la estructura de control if() encerramos entre paréntesis la condición si “aa” es mayor que 14, en este caso devuelve un valor lógico de falso o verdadero, como en este caso “aa” que se interpreta dentro de los paréntesis como 15 la función hace una comparación de esta manera:

* (15 es mayor que 14) = VERDADERO

Seguidamente utilizamos la función de print que se ejecutara si el valor es igual a verdadero imprimiendo lo contenido en sus paréntesis en este caso “SI MAYOR”.

De está manera con las estructuras de control, podemos coaccionar para la toma de decisiones y tomar una ruta determinada dependiendo del resultado de esta.

**Instrucción Compuesta**

Cuando deseamos agregar mas de un resultado a determinada ejecución podemos usar una instrucción compuesta la cual encerraremos entre paréntesis {} de esta forma:

**if(**aa **>** 14**){** #Instruccion compuesta

print**(**"Primer Reglon"**)**

print**(**"Si Mayor"**)**

**}**

En el ejemplo anterior usamos parte del código anterior sin embargo a la respuesta que es verdadera le asignamos una respuesta compuesta de dos notificaciones como lo podemos comprobar ejecutándolo en nuestros computadores.

En el código siguiente vamos a capitalizar la respuesta de esta forma:

y **<-** 10

z **<-** **if(**aa **>** 14**)** 50

z

Arriba asignamos al objeto z el valor de la estructura de control antes asignada.

Ahora vamos a ponerla un poquito mas compleja, ya que utilizaremos varios operadores y expresiones de una forma compuesta.

**if(**10 **>** aa**)** **{** #1mer Bloque

print**(**"Rango Menor"**)**

**}** **else** **if** **(**10 **<=** aa **&&** aa **<=** 20**)** **{** #Segundo Bloque

print**(**"Primer Renglon"**)**; print**(**"Rango Medio"**)**

**}** **else** **{** #Tercer Bloque

print**(**"Rango Mayor"**)**

**}**

Recordemos que el valor asignado a “aa” es igual a 15, entramos a la lógica y nos indica lo siguiente:

Si 10 es mayor que 15 imprima “Rango Menor”, aquí no cumple una condición verdadera, sino “FALSO”, por lo cual recorre a la siguiente estructura y verifica si 10 es menor o igual que 15 y 15 es menor o igual a 20 en donde retorna “VERDADERO”, lo cual activa la impresión de ese bloque, y en caso de que no se cumpla el primero o el segundo ejecutara la tercera condición. Los algoritmos de clasificación funcionan con este principio, pero con una lógica y matemática avanzada, para obtener algún resultado, a este tipo de código siempre es conveniente dejarle una salida en caso de que cualquiera de las condiciones sea falsas.

**Los Ciclos**

En R encontraremos varios tipos de funciones de ciclos o repeticiones, pudiendo ser que ser repitan los mismos un numero determinado de veces ya prestablecido, cuando se cumpla o se deje de cumplir alguna condición o infinitas veces, en este ultimo hay que prestar atención y cuidado ya que fácilmente pueden ocasionar un ciclo infinito en su computador y con ello bloquear memoria y procesadores.

Comenzaremos con repeticiones con un numero determinado de veces; Con repeticiones explicitas (es decir le asignaremos el valor de las repeticiones).

letras **<-** c**(**"c", "L", "i", "M", "T", "A"**)**

**for** **(**i **in** 1**:**6**)** **{**

print**(**letras**[**i**])**

**}**

**Pregunta rápida:**

¿Qué tipo es el objeto letras?

**Su respuesta aquí: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Continuamos….**

En el código anterior, primero le asignamos al objeto letras los valores "c", "L", "i", "M", "T", "A" con la función de contenedor vista anteriormente.

Posteriormente usamos la función for() para ingresar un ciclo de ejecución, dentro de la función cargamos los argumentos i = el cual será un numero aleatorio que se sumará mas uno en lo que recorre el segundo argumento que es 1:6, es decir desde 1 al 6, recordemos que los dos puntos (:) nos sirven para mostrar un rango especifico de números continuos.

La función print llama al objeto letras (de allí la pregunta que tipo de objeto es), y usamos los [] para recorrer la posición de cada uno de los valores contenidos en ese objeto.

Nuestro resultado cuando corramos el código será el siguiente:

[1] "c"

[1] "L"

[1] "i"

[1] "M"

[1] "T"

[1] "A"

Podemos usar también código en donde el numero de veces que se repita cierto ciclo quede implícito; En este caso usaremos la función seq\_along() la cual nos va a generar una secuencia de enteros acorde al numero de elementos que contenga el objeto.

El código quedaría así:

letras **<-** c**(**"c", "L", "i", "M", "T", "A"**)**

**for** **(**i **in** seq\_along**(**letras**))** **{**

print**(**letras**[**i**])**

**}**

En el código anterior vemos que no especificamos el numero de veces, ya que lo tomamos directamente del objeto. Esta técnica la podemos usar cuando asignamos un objeto que se puede variar en su longitud, por ejemplo: Si usamos un objeto que se llame nits de una base de contribuyentes y lo estemos migrando de forma secuencial mes a mes, el objeto se puede llamar igual, pero el contenido puede variar en numero dependiendo de la temporalidad de este.

Al usar la función seq\_along, esta leerá la longitud y asignará el valor de forma automática.

Podemos crear un código mas simplificado y hacerlo aún más implícito, aprovechando la propia esencia de la función for() creando una construcción que tratara de ir tomando cada uno de los elementos del objeto consignado.

letras **<-** c**(**"c", "L", "i", "M", "T", "A"**)**

**for** **(**letra **in** letras**)** **{**

print**(**letra**)**

**}**

**Repeticiones con condición**

Podemos habilitar esta operación de ciclo con la función while(), sin embargo con ella debemos de tener mucho cuidado ya que tenemos si o si, que poner un objeto que indique un índice de entrada y salida ya que si no lo hacemos adecuadamente ocasionamos un ciclo infinito y sin salida.

Para su tranquilidad estos temas los estamos viendo como parte de cultura de programación, R posee estructuras y funciones avanzadas que nos ayudan de una manera más fácil y limpia cuando se necesitan este tipo de trabajos, estos los veremos en capítulos posteriores, aunque si es necesario que veamos sus bases con antelación.

En el siguiente código vemos cómo funciona un ciclo con repeticiones por condición:

letras **<-** c**(**"c", "L", "i", "M", "T", "A"**)**

i **<-** 1

**while(**i **<=** 6**)** **{**

print**(**letras**[**i**])**

i **<-** i **+** 1

**}**

En esta parte de código nuevamente agregamos el objeto letras como en los ejemplos anteriores.

Le asignamos al objeto “i” un valor inicial, en este caso 1, dentro de la función while() estamos especificando que el ciclo continue mientras i sea menor o igual a 6 con el operador lógico “menor o igual que” **“<=”** , esto quiere decir que mientras no se de este caso el algoritmo se repetirá hasta que la condición se cumpla.

En el ciclo imprimirá cada vez que pase el valor que se encuentra en la posición del vector en especifico en este caso el vector se llama letras y se accede a cada una de sus posiciones por el mismo objeto i, siendo por ejemplo lo siguiente.

Letras [1] = “c”, letras[2] = “L” y así sucesivamente hasta que el ciclo se interrumpa.

En el caso que ya no existiera mas contenido en el vector, R le asignara NA a los siguientes valores. Para comprenderlo mejor podemos modificar el valor de “i” cambiándolo de 6 a 8 o 10 o el numero de repeticiones que usted guste.

**Repeticiones Infinitas**

Existe la función repeat() que realiza ciclos infinitos, el lenguaje posee facilidades para poder interrumpir el ciclo desde su interior cuando se cumpla una condición, lo cual veremos en el siguiente código:

letras **<-** c**(**"c", "L", "i", "M", "T", "A"**)**

**repeat** **{**

print**(**letras**[**i**])**

i **<-** i **+** 1

**if(**i **>** 6**)**

**break**

**}**

En este código a diferencia de los otros, declaramos el objeto contador dentro de la función y posteriormente con la función if(), marcamos un alto o parada para la misma disparando la función break.

Los ciclos pueden ser interrumpidos con 3 instrucciones diferentes, ya sea break, next o return.

En el siguiente ejemplo vamos a usar un generador de numero aleatorios para tratar de explicar con claridad estos eventos.

set.seed**(**140**)** #Cualquier numero

aprox **<-** 0.003 #Valor de la salida del ciclo

Y\_ini **<-** 2.7 #Valor inicial de Y supuesto

**for(**iter **in** 1**:**1000**)** **{** #Aseguro no mas de 1000 interaciones

#Procedimiento para calcular la siguiente Y, que en este caso simularemos mediante un generador aleatorio

Y **<-** Y\_ini **+** 0.008**\***rnorm**(**1**)**

**if(**abs**(**Y **-** Y\_ini**)** **<=** aprox**)**

**break** #salir del ciclo

Y\_ini **<-** Y

**}**

paste**(**"Y\_ini", Y\_ini, "Y:", Y, "Num.Iter", iter**)** #Unir el Resultado

Pasaremos a explicar el código:

La función set.seed pone una semilla que tomará como un punto aleatorio para que el ejemplo se comporte igual en todos los equipos, es decir que fuerza que un resultado sea el mismo independientemente del equipo en el que se esté corriendo, aunque sea un trabajo aleatorio.

El objeto “aprox” únicamente nos sirve para establecer el valor de la salida del ciclo, en este caso 0.003

Y\_ini es la ponderación que le damos al objeto como salida 2.7

En la secuencia del for() establecemos un objeto iter que será recorrido no mas de mil veces.

Al objeto “Y” le aplicamos una fórmula matemática en donde tenemos el valor inicial de Y\_ini que es 2.7 mas 0.008 \* rnorm(1), rnorm es una función que nos ayuda a mantener una distribución normal, existe el dnorm, pnorm, qnorm y rnorm, estos los veremos cuando entremos de lleno al área de estadística, pero la función rnorm nos va a ayudar a mantener el ejemplo dentro de una distribución normal con su media y si desviación estándar adecuadamente.

La función abs() también la vemos y nos ayuda a calcular el valor absoluto de lo que estamos obteniendo de los objetos Y restándole el Y\_ini y cuando este sea menor o igual al objeto “aprox” para el ciclo, pasandole el valor a Y de lo que en ese momento tenga de Y\_ini.

Posteriormente usamos la función de paste() que nos ayuda a concatenar los resultados de los objetos que queremos mostrar.

Este solo es un ejemplo de como se puede codificar un ciclo, donde podemos ver que el objetivo fue alcanzado en 8 iteraciones.

En el siguiente ejemplo utilizaremos next para interrumpir flujo normal de la ejecución de una manera diferente, en vez de salir del ciclo, solamente impedirá la ejecución de las instrucciones siguientes, retornando al principio del ciclo nuevamente.

Aquí el código:

**for(**i **in** 1**:**7**)** **{**

**if(**3 **<=** i **&&** i **<=** 5**)**

**next**

print**(**i**)**

**}**