Fuente: www.appinventor.org
Traducción hecha con Google Traductor y mejorada por mi: piatticarlos@gmail.com

Bloques Repetitivos: Iteración

Una cosa para la que los ordenadores son buenos es la repetición de operaciones, igual que los niños pequeños, nunca se cansan de repetir. Son también muy rápidos y puede realizar cosas como procesar en un microsegundo toda su lista de amigos de Facebook.

En este capítulo, aprenderá a programar la repetición con unos pocos bloques en lugar de copiar y pegar los mismos bloques una y otra vez. Va a aprender a hacer cosas como enviar un SMS a cualquier número de teléfono de una lista y ordenar los elementos de la lista. También aprenderá que repetir bloques puede simplificar significativamente el funcionamiento de una aplicación.

Controlar la ejecución de una aplicación: Bifurcaciones y bucles
En los capítulos anteriores, ha aprendido que el comportamiento de una aplicación se define con un conjunto de controladores de eventos: eventos y funciones que responden ejecutandosé. También aprendimos que la respuesta a un evento no es a menudo una secuencia lineal de funciones y puede contener bloques que se realizan sólo bajo ciertas condiciones.

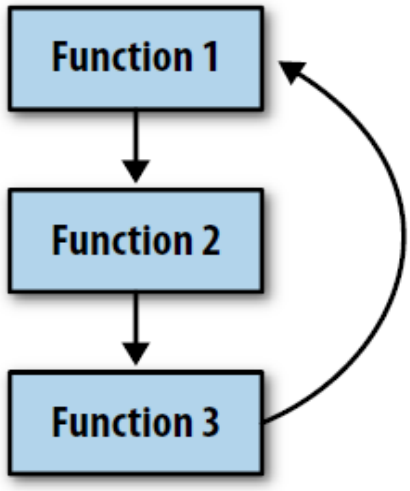

Figure 20-1. Repeat blocks cause a program to loop

La otra forma en que se comporta una aplicación no lineal es mediante la repetición de bloques. Como los bloques **if** y **Ifelse** permiten la ramificación de un programa, los bloques de repetición hacen que un programa realice bucles, es decir, llevar a cabo un conjunto de funciones y luego saltar hacia atras en el código y hacerlo nuevamente, como se ilustra en la Figura 20-1.

Cuando se ejecuta una aplicación, hay un contador de programa que trabaja ocultamente en la aplicación realizando un seguimiento de la siguiente operación a realizarse. Hasta ahora, usted ha examinado las aplicaciones en las que el contador de programa se inicia en la parte superior de un controlador de eventos y (condicionalmente) realiza las operaciones desde arriba hacia abajo. Con los bloques de repetición, el contador de programa regresa en los bloques, repitiendo funciones continuamente
.

En App Inventor, hay dos tipos de bloques de repetición: **foreach** y **while.foreach** que se utilizan para especificar las funciones que se deben realizar a cada elemento de una lista. Por lo cual, si usted tiene una lista de números de teléfono, puede especificar que se debe enviar un texto a cada número de la lista.

El bloque **while** es más general que el **foreach**. Con él, usted puede programar que los bloques se repitan continuamente hasta que cambie alguna condición arbitraria. Los bloques while se pueden utilizar para calcular fórmulas matemáticas, tales como la adición del primer número *n* o calcular el factorial de *n*. También puede utilizar while cuando necesita procesar dos listas simultáneamente; **foreach** procesa solamente una lista a la vez.

Funciones de repetición en una lista utilizando foreach

En el capítulo 18, hablamos de la aplicación Random Call. Llamar a un amigo aleatoriamente a veces podría funcionar, pero si usted tiene amigos como los míos, no siempre le responderán. Una estrategia diferente sería enviar un texto "Missing you" (Te desapareciste) a todos tus amigos y ver quién responde primero.

En esta aplicación, al hacer clic en un botón, se envía un SMS a más de un amigo. Una forma de implementar esto sería simplemente copiar los bloques para enviar mensajes de texto a un solo número, y luego copiar y pegar para cada amigo que desee enviarle el texto, como se muestra en la Figura 20-2.

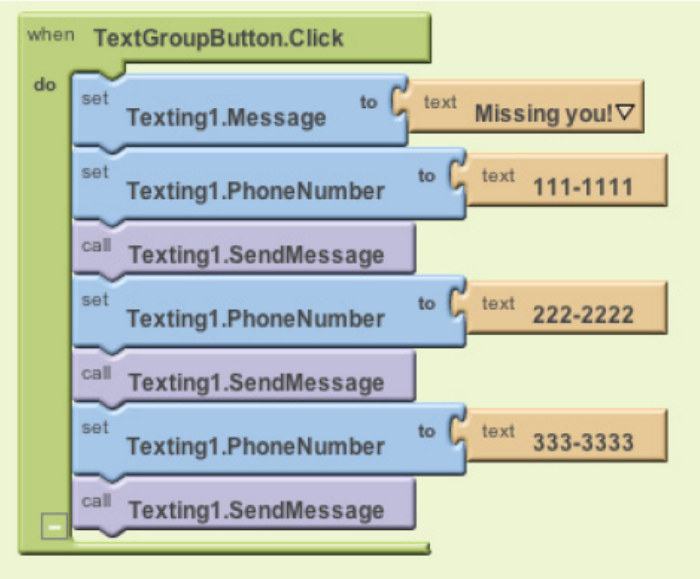

Figure 20-2. Copying and pasting the blocks for each phone number to be texted

Este método de "fuerza bruta" copy-paste está bien si usted tiene que repetir unos pocos bloques. Pero las listas de datos, como la lista de sus amigos, tienden a cambiar. Usted no quiere modificar su aplicación con el método de copiar y pegar cada vez que agregue o elimine un número de teléfono de su lista.

El bloque **foreach** proporciona una solución mejor. Se define una variable de lista PhoneNumbers con todos los números y luego se conforma un bloque **foreach** en torno a una única copia de los bloques que se desean realizar. La Figura 20-3 muestra la solución **foreach** para enviar mensajes de texto a un grupo.

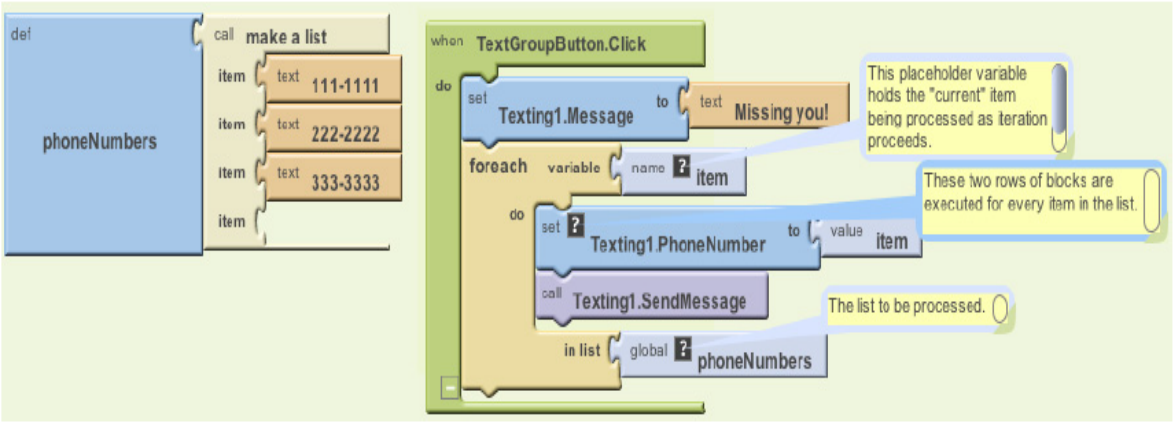

Figure 20-3. Using the foreach block to perform the same blocks for each item in the list

1° nota en la imagen:

Esta variable contiene el marcador de posición "actual" del item que se está procesando a medida que avanza la iteración.
2° nota en la imagen:
Estas dos filas de bloques se ejecutan para cada item de la lista.
3° nota en la imagen:
La lista donde se realizará el proceso.

Al arrastrar un bloque **foreach**, debe especificar la lista para procesar conectando una referencia al parámetro "in list" en la parte inferior del bloque. En este caso, el bloque **global phoneNumbers** fue sacado de la sección My Definitions y conectado para proporcionar la lista de números de teléfono a los cuales enviarles texto.

En la parte superior del bloque **foreach**, también se introduce un nombre para la variable de *señalamiento* (marcador de posición) que viene con **foreach**. Por defecto, este marcador de posición se denomina "var." Puedes dejarlo así o cambiarle el nombre. Un nombre común para esto es "item", ya que representa el elemento que se está procesando en la lista.

Los bloques dentro de **foreach** se repiten para cada item de la lista, con la variable de señalamiento (en este ejemplo, ítem) manteniendo siempre el elemento que se está procesando. Si una lista tiene tres elementos, los bloques interiores se ejecutarán en tres ocasiones. Se dice que los bloques internos están subordinados al bloque **foreach**. El contador de programa “hace looping” (vuelve a repetir) desde el principio cuando llega a la parte inferior dentro del bloque **foreach**.

Más detalles acerca de Looping
Vamos a examinar la mecánica de los bloques **foreach** en detalle, porque comprender los loops es fundamental para la programación. Cuando se hace clic en el botón TextGroupButton y se invoca al controlador de eventos, la primera operación ejecutada es el bloque **set Texting1.Message**, que ajusta el mensaje "Missing you." Este bloque sólo se ejecuta una vez.

Entonces el bloque **foreach** comienza antes de que los bloques internos se ejecuten, la variable de señalamiento item se establece en el primer número de la lista PhoneNumbers (111-1111). Esto sucede automáticamente; **foreach** le libera de tener que llamar manualmente a **select list item**. Después de que el primer elemento es seleccionado en la variable item, los bloques dentro de **foreach** se ejecutan por primera vez. La propiedad Texting1.PhoneNumber se establece en el valor del item (111-1111), y el mensaje es enviado.

Después de alcanzar el último bloque dentro de **foreach** (**Texting.SendMessage**), la aplicación hace "looping" nuevamente al comienzo de los bloques dentro de foreach y automáticamente pone el siguiente item de la lista (222-2222) en la variable *item*. Las dos operaciones dentro del foreach se repiten, el envío del texto "Missing you" al 222 2222. Luego la aplicación vuelve nuevamente y establece la variable *item* para el último elemento de la lista (333-3333). Las operaciones se repitieron una tercera vez, para enviar el tercer texto.

Debido a que el elemento final de la lista (en este caso, la tercera) ha sido procesado, el bucle **foreach** se detiene en este punto. Se dice que el control “saltó” fuera del bucle, lo que significa que el contador de programa se dispone a seguir con los bloques debajo de **foreach**. En este ejemplo, no hay bloques por debajo de él, por lo tanto el controlador de eventos llega a su fin.

Escribir un Código sustentable

Para el usuario final, la solución **foreach** que acabamos de describir se comporta exactamente como el método de "fuerza bruta" copiar y luego pegar los bloques de mensajes de texto. Desde la perspectiva de un programador, sin embargo, la solución **foreach** es más fácil de mantener y se puede utilizar incluso si los datos (la lista de teléfonos) se introducen de forma dinámica.

El software sustentable es un software que se puede modificar fácilmente sin introducir errores. Con la solución de **foreach**, puede cambiar la lista de amigos a las que se envían textos, modificando únicamente la lista de variables *“no es necesario cambiar la lógica del programa (el controlador de eventos) en absoluto”*. Esto contrasta con el método de fuerza bruta, lo que requiere que se añadan nuevos bloques en el controlador de eventos cuando se añade un contacto nuevo. Cada vez que modifique la lógica de un programa, corre el riesgo de introducir errores.

Aún más importante, la solución **foreach** funcionará incluso si la lista de teléfonos fuese una lista dinámica, es decir, aquella en la que el usuario final, no sólo el programador, puede añadir números a la lista. A diferencia del ejemplo, que cuenta con tres números de teléfono listados en el código, la mayoría de las aplicaciones trabajan con datos dinámicos que vienen del usuario final o alguna otra fuente. Si rediseña esta aplicación para que el usuario final pueda introducir los números de teléfono, usted tendría que usar la solución **foreach**, porque cuando se escribe el programa, no se sabe los números que hay que poner en el método de fuerza bruta.

Un segundo ejemplo de Foreach: Mostrar una Lista

Cuando desee mostrar los elementos de una lista en el teléfono, puede conectar la lista dentro de la propiedad *Text* de un *Label*, como se muestra en la Figura 20-4.

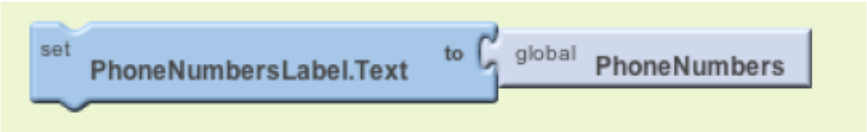

Figure 20-4. The simple way to display a list is to plug it directly into a label

Cuando se conecta una lista directamente en una propiedad Text de un control Label, los elementos de la lista se muestran en la etiqueta como una sola fila de texto separado por espacios y contenidos en paréntesis:

(111–1111 222–2222 333–3333)

Los números pueden, o no, ocupar más de una línea, dependiendo de cuántos hay. El usuario puede ver los datos y tal vez comprender que se trata de una lista de números de teléfono, pero no es una manera muy elegante. Los elementos de una lista normalmente se muestran en líneas separadas o separados con comas.

Para mostrar una lista correctamente, es necesario transformar los bloques de cada item de la lista en un valor solamente de texto con el formato que desee. Los objetos de texto en general están formados por letras, dígitos y signos de puntuación. Pero el texto también puede almacenar caracteres especiales de control, que no se asignan a un caracter visible. Un fichero, por ejemplo, se denota por \ t. (Para obtener más información sobre los caracteres de control, revisar el estándar Unicode para la representación de texto en http://www.unicode.org/standard/standard.html.)

En nuestra lista de números de teléfono, deseamos colocar un carácter de nueva línea, que se denota por \n. Entonces si \n aparece en un bloque de texto, esto significa "ir a la siguiente línea antes de mostrar lo siguiente" Así que el objeto de texto "111-1111\n222-2222\n333-3333" aparecería así:

111–1111
222–2222
333–3333

Para construir un objeto de texto, utilizaremos un bloque **foreach** y "procesaremos" cada item añadiendo la propiedad PhoneNumberLabel.text y un carácter de nueva línea a la propiedad PhoneNumberLabel.Text, como se muestra en la Figura 20-5.

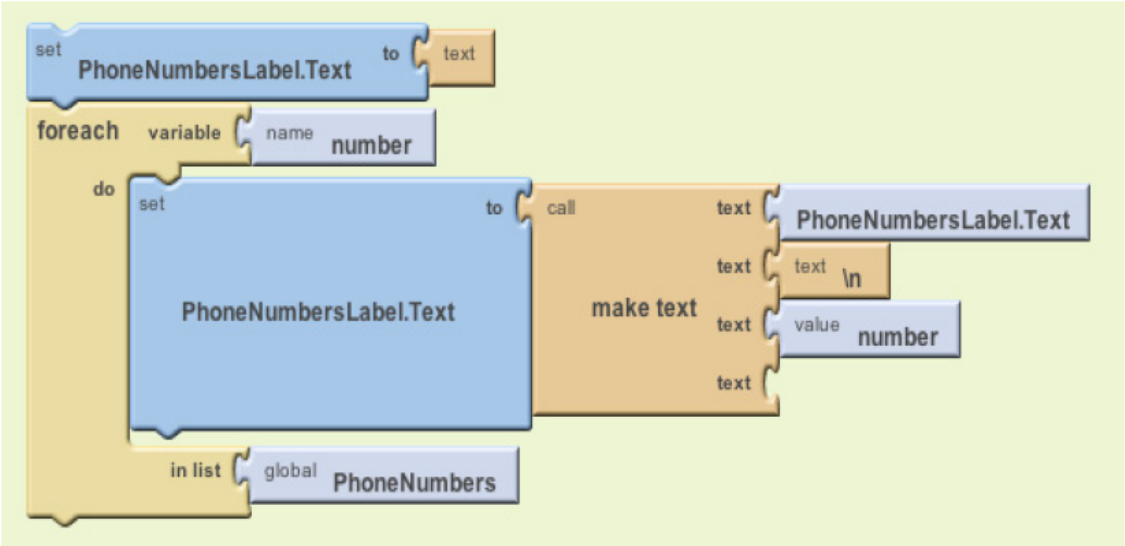

Figure 20-5. Using the foreach block to process the list and put a newline character before each item

Vamos a rastrear los bloques para ver cómo funcionan. Como se discutió en el capítulo 15, el seguimiento muestra cómo cambia cada propiedad o variable cuando los bloques se ejecutan. Con **foreach**, consideramos los valores después de cada *iteración*, es decir, cada vez que el programa pasa por el bucle **foreach**.

Antes de **foreach**, el PhoneNumbersLabel se inicializa con el texto vacío. Cuando **foreach** comienza, la aplicación automáticamente ubica el primer elemento de la lista (111-1111) en la variable de marcador de posición **number**. Los bloques en **foreach** y luego **make text** con **PhoneNumbersLabel.Text** (texto vacío), \n, y **number**, establecen el resultado en **PhoneNumbersLabel.Text**. Así, después de la primera repetición de **foreach**, las variables pertinentes almacenan los valores mostrados en la Tabla 20-1.

Table 20-1. The values of the variables after the first iteration of foreach

number	PhoneNumbersLabel.Text
111–1111	\n111–1111

Ya que la parte inferior de **foreach** se ha alcanzado, el control vuelve nuevamente hacia arriba y el siguiente elemento de la lista (222-2222) se pone en la variable **number**. Cuando los bloques interiores se repiten, **make text** concatena el valor de **PhoneNumbersLabel.Text** (\n111-1111) con \n, y luego con **number**, que ahora es 222-2222. Después de esta segunda iteración, las variables almacenan los valores que se muestran en la Tabla 20-2.

Table 20-2. The variable values after the second iteration of foreach

number	PhoneNumbersLabel.Text
222–2222	\n111–1111\n222–2222

entonces el tercer item de la lista se coloca en **number**, y el bloque interno se repite una tercera vez. El valor final de las variables, después de esta última iteración, se muestran en la Tabla 20-3.

Table 20-3. The variable values after the final iteration

number	PhoneNumbersLabel.Text
333–3333	\n111–1111\n222–2222\n333–3333

Así, después de cada iteración, *label* se hace más grande y tiene un número de teléfono más (y una nueva línea más). Al finalizar **foreach**, PhoneNumbersLabel.Text se establece de modo que los números aparecerán de la siguiente manera:

111–1111
222–2222
333–3333

Repetición de bloques con while

El bloque **while** (siempre que) es un poco más complicado de usar que **foreach**. La ventaja del bloque **while** reside en su generalidad: **foreach** repite sobre una lista, pero **while** puede repetir *siempre que una condición arbitraria sea verdadera*. Como un ejemplo trivial, suponga que desea enviar el texto a cualquier otra persona en su lista de teléfonos. No podía hacerlo con **foreach**, pero con **while**, podría incrementar el índice de a dos en vez de uno por vez.

Como se vio en el capítulo 18, una condición comprueba algo y devuelve un valor verdadero o falso. Los bloques **while-do** contienen una prueba condicional, al igual que los bloques **if**. Si un bloque **while** en su comprobación el resultado de la evaluación es *true* “verdadero”, la aplicación ejecuta los bloques interiores, regresa nuevamente y vuelve a verificar la comprobación. Siempre y cuando la prueba se evalúe como *true* “verdadero”, los bloques interiores se repitrán. Cuando la prueba se evalúa como *false* “falso”, la aplicación "salta" fuera bucle (como vimos en el bloque **foreach**) y continúa con los bloques por debajo de **while**.

Usando while para procesar sincrónicamente dos listas

Un ejemplo más instructivo de **while** y su generalidad trata de situaciones en las que se necesitan procesar dos listas de forma sincrónica. Por ejemplo, en la aplicación MakeQuiz (capítulo 10), se mantienen listas separadas de las preguntas del examen y respuestas, junto con una variable `index` para realizar un seguimiento del número de la pregunta actual. Para mostrar cada par pregunta-respuesta en conjunto, es necesario iterar a través de las dos listas de forma sincrónica, tomando el artículo indexth de cada uno. **foreach** permite solamente recorrer una única lista, pero con un bucle **while**, se puede utilizar el index para agarrar un elemento de cada lista. La figura 20-6 ilustra el uso de un bloque **while** para mostrar los pares pregunta-respuesta en líneas separadas.

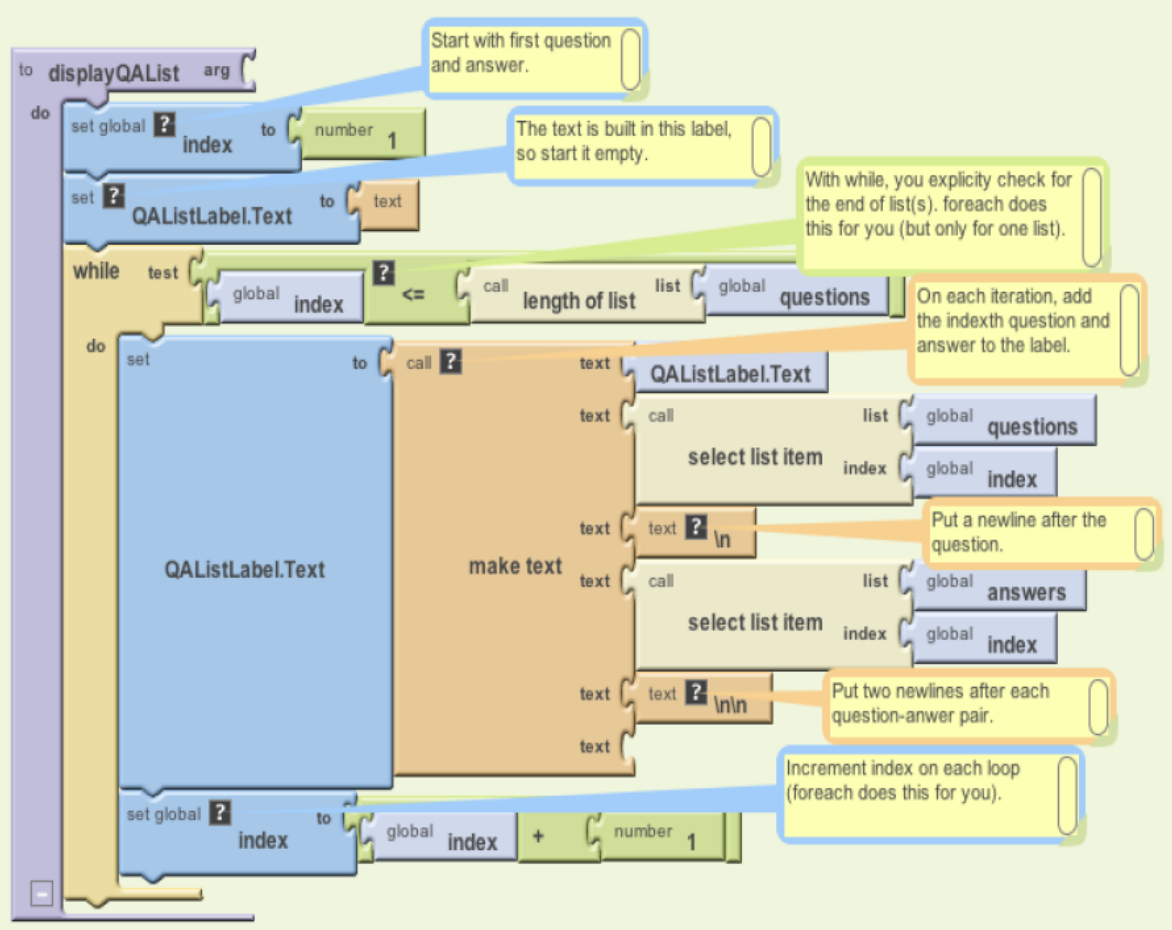

Figure 20-6. Using a while loop to display the question-answer pairs on separate lines

1° nota en la imagen: Comience con la primer pregunta y respuesta
2° nota en la imagen: El texto se basa en esta etiqueta, así que empieza vacía
3° nota en la imagen: Con <while>, usted le indica para comprobar el fin de la lista(s). <foreach> hace esto por usted (pero sólo para una lista).
4° nota en la imagen: En cada repetición, añadir el indexth de pregunta y respuesta a la etiqueta
5° nota en la imagen: Poner una nueva línea después de la pregunta
6° nota en la imagen: Ponga dos saltos de línea después de cada par de pregunta-respuesta
7° nota en la imagen: Incrementar el índice en cada bucle (foreach lo hace por usted)

Debido a que se utiliza **while** en lugar de **foreach**, explícitamente hay que introducir los bloques que inicializan el índice, buscan el final de la lista, seleccionan los elementos de cada bucle, e incrementan el índice.

Uso de while para calcular una fórmula

He aquí otro ejemplo de **while** que repite las operaciones, pero no tiene nada que ver con una lista. ¿Qué cree usted que realizan los bloques de la Figura 20-7 en un alto nivel? Una forma de resolver esto es rastrear cada bloque (véase el capítulo 15 para más información sobre el rastreo), el seguimiento del valor de cada variable sobre la marcha.

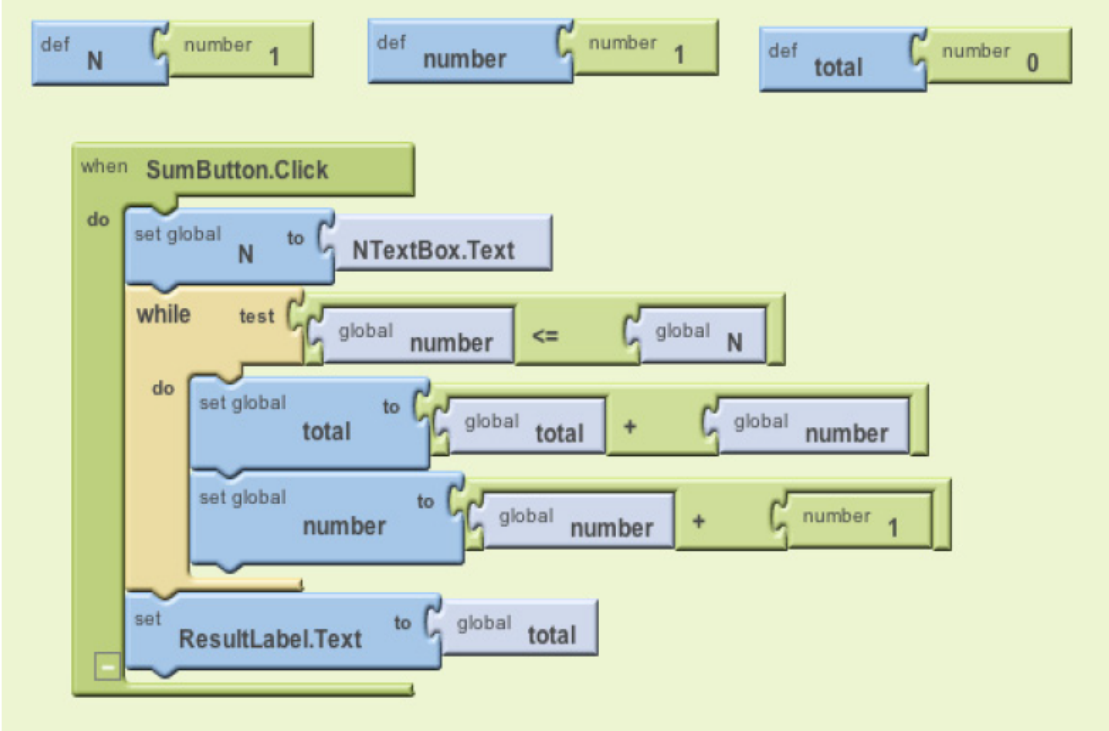

Figure 20-7. Can you figure out what these blocks are doing?

Los bloques dentro del bucle **while** se repiten *mientras que la variable number sea menor o igual que la variable N*. En esta aplicación, N se establece en el número que el usuario ingrese en el cuadro de texto (NTextBox). Digamos que el usuario ha introducido un 3. Las variables de la aplicación se verían como en la tabla 20-4 cuando el bloque **while** se alcance.

Table 20-4. This is how the variables look when the while block is reached

N	number	total
3	1	0

El bloque **while** primero pregunta: ¿number es menor o igual que (**<=**) N? La primera vez que se hace esta pregunta, la prueba es verdadera, por lo que continúa la ejecución del bloque

while. El total se establece a sí mismo (0), + number (1), y number se incrementa. Después de la primera iteración de los bloques dentro de **while**, los valores de las variables son como se enumeran en la Tabla 20-5.

Table 20-5. The variable values after the first iteration of the blocks within the while block

N	number	total
3	2	1

En la segunda iteración, la prueba "number <= N" sigue siendo verdadero (2 <= 3), de modo que los bloques internos se ejecutan de nuevo. total se fija a sí mismo (1) + number (2). number se incrementa. Cuando esta segunda iteración se completa, las variables quedan como se muestran en la Tabla 20-6.

Table 20-6. The variable values after the second iteration

N	number	total
3	3	3

La aplicación repite nuevamente y prueba la condición. Una vez más, es verdadero (3 <= 3), de modo que los bloques son ejecutados una tercera vez. Ahora total se establece a sí mismo (3) + number (3), por lo que se convierte en 6. number se incrementa a 4, como se muestra en la Tabla 20-7.

Table 20-7. The values after the third iteration

N	number	total
3	4	6

Después de esta tercera iteración, el control de bucle repite todo una vez más. Ahora la prueba "number <= N", o “4 <= 3”, se evalúa como falsa. Por lo cual los bloques internos de **while** no se ejecutan de nuevo, y el controlador de eventos llega a su fin

Entonces, ¿qué hacen estos bloques? Llevaron a cabo una de las operaciones matemáticas más fundamentales: contar números. Cualquier número que el usuario introduce, la aplicación notificará la suma de los números 1 .. N, donde N es el número introducido. En este ejemplo, asumimos que el usuario ha introducido 3, por lo que la aplicación se acercó con un total de 6. Si el usuario hubiera introducido 4, la aplicación llegaría a 10..

Resumen

Los ordenadores son buenos para la repetición de la misma función una y otra vez. Imagine todas las cuentas bancarias que se procesan para devengar intereses, todos los grados

procesados para calcular los promedios de calificaciones de los estudiantes, y un sinnúmero de otros ejemplos cotidianos donde las computadoras utilizan la repetición para realizar una tarea.

App Inventor proporciona dos bloques para repetir operaciones. El bloque **foreach** aplica un conjunto de funciones para cada elemento de una lista. Con su utilización, usted puede diseñar un código de procesamiento que funciona en una lista abstracta en lugar de datos concretos. Este código es más fácil de mantener, y es muy util si los datos son dinámicos.

En comparación con **foreach**, **while** es más general: se puede utilizar para procesar una lista, pero también se puede utilizar para procesar sincrónicamente dos listas o calcular una fórmula. Con **while**, los bloques interiores se realizan de forma continua mientras que una cierta condición sea verdadera. Después que se ejecutan los bloques dentro de while, el control vuelve de nuevo hacia arriba y nuevamente se realiza la prueba de condición. El bloque **while** llega a su fin solamente cuando la prueba se evalúa como falsa.

Fuente: www.appinventor.org
Traducción hecha con Google Traductor y mejorada por mi: piatticarlos@gmail.com