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

Programación de Listas de Datos

Como usted ya ha visto, la gestión de eventos y la toma de decisiones en las aplicaciones es fundamental para la computación. Pero la otra parte fundamental de una aplicación son sus datos (cuando la información es procesada). Una aplicación de datos rara vez se limita a memorizaciones simples tales como el resultado de un juego. Por lo general, se trata de elementos complejos, interrelacionados que deben organizarse con atención a la funcionalidad de la aplicación.

En este capítulo, vamos a examinar la forma en que App Inventor maneja los datos. Usted aprenderá los fundamentos de la programación de dos listas: las listas estáticas (en la que los datos no cambian) y las listas dinámicas (en la que los datos son generados por los usuarios). A continuación, usted aprenderá cómo hacer frente a datos aún más complejos todavía, que involucran listas cuyos elementos también son listas.

Muchas aplicaciones procesan listas de datos. Por ejemplo, Facebook procesa su lista de amigos. La aplicación Quiz President trabaja con una lista de preguntas y respuestas. Un juego puede tener una lista de caracteres o una de todas las fechas con las puntuaciones más altas.

Las variables *List* funcionan como las variables de texto y las variables de número como en las que hemos trabajado, pero en lugar de representar una sola celda de memoria, representa un conjunto de celdas de memoria. Consideremos, por ejemplo, la lista de números de teléfono en la Tabla 19-1.

Table 19-1. A list variable represents a set of memory cells

111–2222
333–4444
555–6666

A los elementos de una lista se accede mediante un índice. Un índice es una posición en una lista, por lo que el índice 1 de la lista en la Tabla 19-1 se refiere al 111-2222, el índice 2 se refiere al 333-4444, y el índice 3 se refiere al 555-6666.

App Inventor proporciona bloques para crear listas, añadir elementos a las listas, seleccionar elementos determinados de una lista, y aplicar operaciones a una lista entera. Comencemos por cómo crear una lista.

Crear una Lista de Variables

Puede crear una lista de variables en el editor de bloques utilizando un bloque **def variable** y un bloque make a list. Por ejemplo, supongamos que usted está haciendo una aplicación para escribir con un solo click los números de teléfono que están en una lista. Usted crea la lista de números de teléfono de la siguiente manera:

1. Desde la sección Built-In, arrastre un bloque **def variable** (Figura 19-1) hacia el área de construcción.

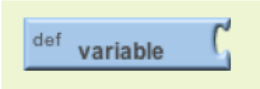

Figure 19-1. A def variable block

2. Haga clic en el texto "variable" y cambie el nombre a "PhoneNumbers", como se muestra en la Figura 19-2.

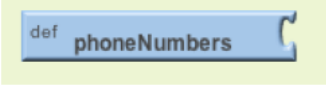

Figure 19-2. Renaming the variable to phoneNumbers

3. Desde la sección Listas, arrastre un bloque **make a list** y conéctelo en el bloque definición, como se muestra en la Figura 19-3. Esto le dice a App Inventor que en la variable debe almacenar una lista de datos en lugar de un único valor.

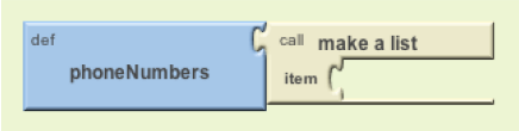

Figure 19-3. Defining phoneNumbers as a list using the make a list block

4. Finalmente, arrastre algunos bloques de texto, introduzca los números de teléfono deseados, y conecte los bloques en las ranuras "item" para conformar un bloque de lista. Tenga en cuenta que cada vez que se agrega un nuevo elemento a la lista se abrirá una nueva ranura “item” en la parte inferior, como se muestra en la Figura 19-4.

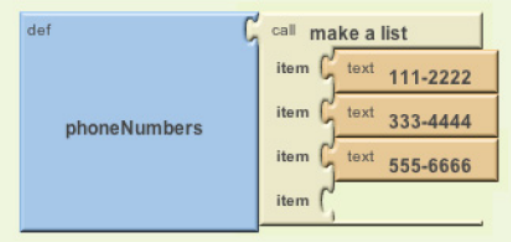

Figure 19-4. As each item is added to the list, a new slot opens up

Usted puede conectar cualquier tipo de datos en una ranura "item", pero en este caso, los items deben ser objetos de texto, y no números, porque los números de teléfono tienen guiones y otros símbolos de formato que no se pueden poner en un objeto de número, y usted no va a realizar ningún cálculo sobre el número (en este caso, usted desea que sean objetos en lugar de números).

Los bloques de la figura 19-4 son para definir una variable llamada PhoneNumbers. Las definiciones de las variables se cargan cuando la aplicación es iniciada, por lo que los slots de memoria como los de la Tabla 19-1 se llenan cuando la aplicación se inicia. Una vez que tenga una lista de variables, es hora de empezar a trabajar con los datos de la lista.

Selección de un elemento de la Lista

Su aplicación puede acceder a los elementos individuales de una lista con el bloque **Select Item list** mediante la especificación de un índice en la lista. El *index* (índice) indica la posición de un elemento dentro de una lista. Por lo tanto, si una lista tiene tres elementos, puede acceder a los elementos con los índices 1, 2 y 3. La figura 19-5 muestra los bloques que seleccionan el segundo elemento de una lista.

Figure 19-5. Selecting the second item of a list

Con **select list item**, usted conecta la lista que desea en la primer ranura, y el índice que desea en la segunda ranura. Los bloques de la Figura 19-5 le indican a la aplicación que seleccione el segundo elemento de la lista PhoneNumbers. Si se selecciona de la lista PhoneNumbers definida en la Tabla 19-1, el resultado sería "333-4444".

Seleccionar un elemento de una lista sólo es un primer paso (una vez que haya seleccionado el elemento, puede hacer una variedad de cosas con él). A continuación vamos a ver algunos ejemplos.

Utilizando un índice para recorrer una lista
En muchas de las aplicaciones, podrás definir una lista de datos y luego permitir al usuario desplazarse por ella. La app Quiz Presidents dell capítulo 8 proporciona un buen ejemplo de ello: en esa aplicación, cuando el usuario hace clic en un botón Siguiente, se selecciona y se muestra el elemento siguiente de una lista de preguntas.

Pero, ¿cómo seleccionar el siguiente elemento de la lista? Nuestro ejemplo en la Figura 19-5 es seleccionado el elemento 2 de PhoneNumbers. Cuando se recorre una lista, el número de artículo que usted está seleccionando cambia cada momento, es relativo a su posición actual en la lista. Por lo tanto, es necesario definir una variable para representar esa posición actual. Index es el nombre más común utilizado para este tipo de variable, y generalmente se inicializa a 1 (la primera posición en la lista), como se muestra en la Figura 19-6.

Figure 19-6. Initializing the variable index to 1

Cuando el usuario hace algo para ir al siguiente elemento, usted *incrementa* la variable index mediante la adición de un valor de 1 a la misma, y luego selecciona de la lista utilizando ese valor aumentado. La figura 19-7 muestra los bloques para hacer esto.

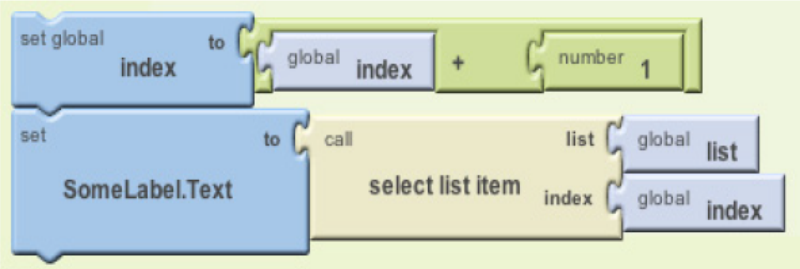

Figure 19-7. Incrementing the index value and using the incremented value to select the next list item

Ejemplo: Desplazamiento de una lista de colores
Echemos un vistazo a una aplicación de ejemplo que le permite al usuario ver detenidamente cada color de pintura posible para su casa, haciendo clic en un botón. Cada vez que se hace clic en el botón, cambia el color de fondo del botón. Cuando el usuario termina de recorrer a través de todos los colores posibles, la aplicación le lleva de vuelta a la primera.

Para este ejemplo, vamos a utilizar algunos colores básicos. Sin embargo, puede personalizar los bloques de código para explorar a través de un conjunto de colores. Para obtener más información sobre colores, consulte la documentación de App Inventor en: http://beta.appinventor.mit.edu/learn/reference/blocks/colors.html

Nuestro primer paso es definir una variable de lista para crear la lista de colores e inicializar con algunos colores en los items, como se muestra en la Figura 19-8.

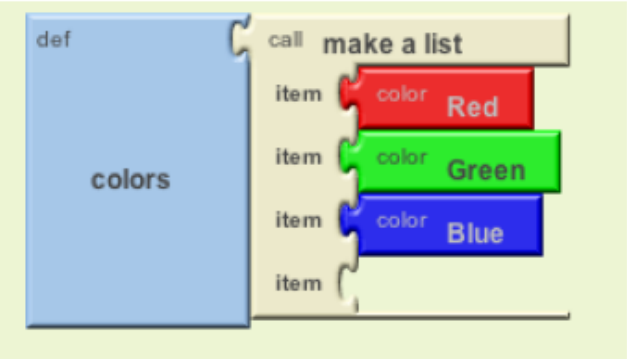

Figure 19-8. Initializing the list colors with a list of paint colors

Luego, definir una variable *index* (índice) que rastrea la posición actual en la lista. Se debe comenzar en 1. Usted podría dar a la variable un nombre descriptivo, como currentColorIndex, pero si no hay muchos índices en su aplicación, se puede nombrar índice, como se muestra en la Figura 19-9.

Figure 19-9. Using the index variable, which is initialized to 1, to track the current position in a list

El usuario puede recorrer por la lista hacia el siguiente item (color) haciendo clic en el botón ColorButton. Cuando se hace clic en él, el índice debe ser incrementado y el BackgroundColor (color de fondo) del botón debería cambiar al elemento actualmente seleccionado, como se muestra en la Figura 19-10.

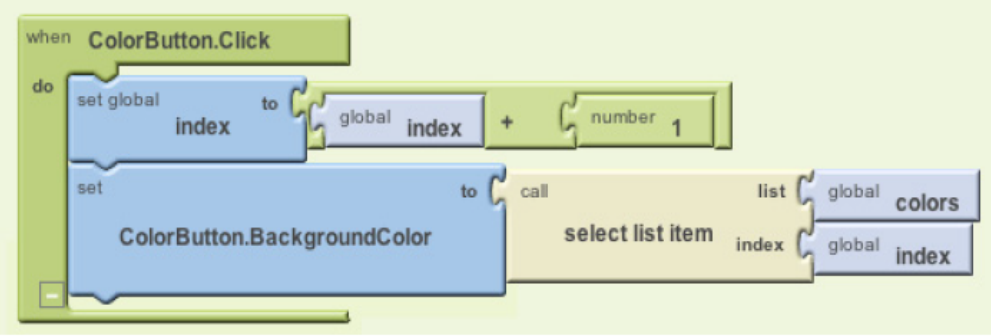

Figure 19-10. Letting the user traverse the color list by clicking a button—changing the button color with each click

Vamos a suponer que en el Diseñador de Componentes el fondo del botón se establece inicialmente en rojo. La primera vez que se pulse el botón, el índice pasará de su valor inicial de 1 a 2, y el color de fondo del botón cambiará al segundo elemento de la lista, Verde. La segunda vez que el usuario hace clic, el índice cambiará de 2 a 3, y el color de fondo se cambiará a azul.

Pero, ¿qué cree que pasará en próximo clic?

Si usted contestó que ocurriría un error, acertó! El `index` (índice) se convertirá en 4 y la aplicación intentará seleccionar el cuarto elemento de la lista, pero la lista sólo tiene tres elementos. La aplicación se cerrará forzosamente, y el usuario verá un mensaje de error como el de la figura 19-11.

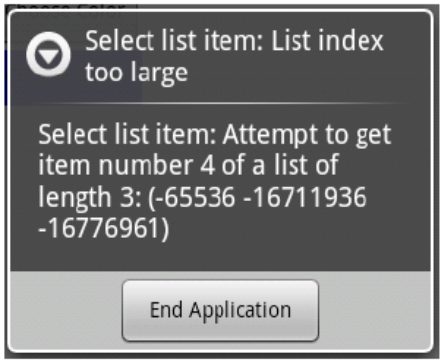

Figure 19-11. The error message displayed when the app tries to select the fourth item from a three-item list

Obviamente, usted no quiere que los usuarios de su aplicación vean este mensaje. Para evitar este problema, agregue un bloque **if** para comprobar si se ha alcanzado el final de la lista. Si es así, el `index` se puede cambiar de nuevo a 1, de modo que el primer color se muestra nuevamente, como se ilustra en la figura 19-12.

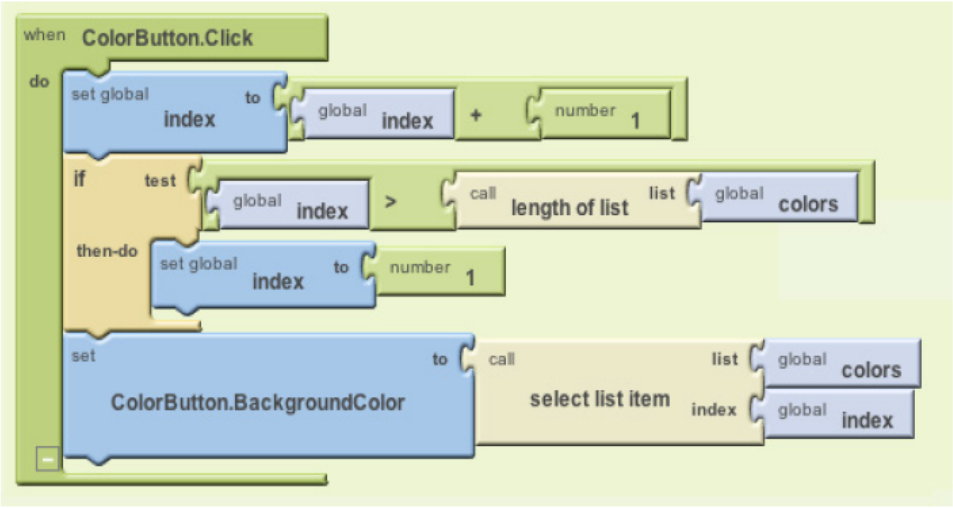

Figure 19-12. Using the if test to check for whether the index value is larger than the length of the list, and reset it to 1 if the test returns true

Cuando el usuario hace clic en el botón, el index se incrementa y luego comprueba para ver si su valor es demasiado alto. El index se compara con **length of list** (longitud de lista), y nó con el número 3, por lo que su aplicación funcionará incluso si se agregan elementos a la lista. Al comprobar si el índice es mayor que la longitud de su lista (en lugar de comprobar si es mayor que el número específico 3), ha eliminado la *dependencia de código* en su aplicación *(code dependency)*. Una *dependencia de código* es un término de programación para casos en los que se programan aspectos muy específicos de su aplicación, de tal manera que si cambia algo en un lugar (por ejemplo, los elementos de la lista), tendrá que perseguir a todos los lugares de su aplicación en los que se utiliza esa lista y cambiar esos bloques también.

Como se puede imaginar, este tipo de dependencias podría causar problemas muy rápidamente, y por lo general llevan a tener que rastrear muchos “bugs”. De hecho, el diseño de nuestra aplicación *“House Paint Color”* contiene otra dependencia de código que tenemos en lo actualmente programado, ¿puede averiguar cual es?

Si ha cambiado el primer color de su lista de Rojo a algún otro color, la aplicación no funcionará correctamente a menos que también haya recordado cambiar el ajuste inicial de Button .BackgroundColor que estableció en el Diseñador de componentes. La manera de eliminar esta dependencia de código es establecer el **ColorButton.BackgroundColor** inicial al primer color de la lista en lugar de ajustarlo a un color específico. Puesto que este cambio implica un comportamiento que se produce cuando su aplicación se abre por primera vez, lo hará en el controlador de eventos **Screen.Initialize** que se invoca cuando se inicia una aplicación, como se ilustra en la Figura 19-13.

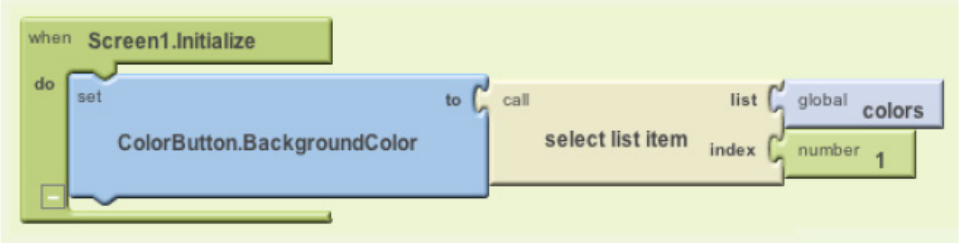

Figure 19-13. Setting the BackgroundColor of the button to the first color in the list when the app is launched

Crear formularios de entrada y listas dinámicas

La aplicación anterior House Paint Color implicó una lista *estática*: una cuyos elementos son definidos por el programador (usted) y cuyos elementos no cambian a menos que se cambien los bloques en el programa. Sin embargo, usualmente, hay que lidiar con aplicaciones usando listas *dinámicas*: listas que cambian en función del usuario que introduce nuevos objetos, o elementos que se cargan desde una base de datos o fuente de información web. En esta sección, discutiremos el ejemplo de la aplicación Note Taker, en la que el usuario introduce notas en un formulario y además se pueden ver las notas anteriores.

Definición de una lista dinámica

Al igual que con una lista estática, a una lista dinámica se la debe definir con el bloque **make a list**. Pero a una lista dinámica, no le agrega ningún item predefinido en la definición de la lista. Por ejemplo, considere la aplicación Note Taker. Podría definir la lista dinámica de notas como en la figura 19-14.

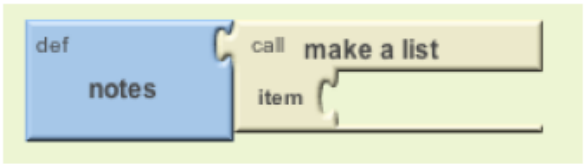

Figure 19-14. The blocks to define a dynamic list don't contain any predefined items

Agregar un "item" (elemento)

La primera vez que alguien pone en marcha la aplicación, la lista de notas está vacía. Sin embargo, cuando el usuario introduce algunos datos en un formulario y hace clic en Enviar, las nuevas notas se añadirán a la lista. La forma puede ser tan simple como la que se muestra en la figura 19-15.

Figure 19-15. Using a form to add new items to the notes list

Cuando el usuario escribe una nota y hace clic en el botón Enviar, la aplicación llama a la función **add items to list** para agregar el nuevo elemento que ingresó en la lista, como se muestra en la Figura 19-16.

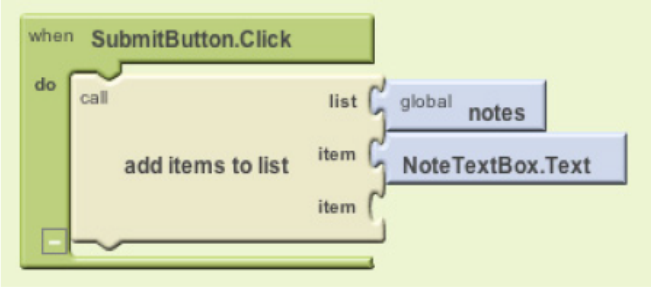

Figure 19-16. Calling add items to list to add the new note when the user clicks the SubmitButton

El bloque **add items to list** agrega el elemento al final de la lista. Cada vez que el usuario hace clic en SubmitButton, se añade una nueva nota.
Usted encontrará el bloque **add items to list** en la sección de Lista. Tenga cuidado: también hay un bloque **append to list**, pero se trata de un bloque bastante raro que sirve para anexar una lista a otra.

Visualización de una lista
El contenido como las notas dentro de las variables de tipo lista no son visibles para el usuario, usted recordará que una variable es un camino para que la aplicación recuerde la información que no necesariamente se muestra al usuario. Los bloques de la figura 19-16 añadirán notas en la lista de artículos en cada clic que se haga en el botón *submit*, pero el usuario no verá ninguna información de lo que está sucediendo hasta que se programen más bloques para mostrar realmente el contenido de la lista.

La forma más sencilla de mostrar una lista en la interfaz de usuario de su aplicación es utilizar el mismo método que se utiliza para mostrar números y texto: poniendo la lista en la propiedad Text de un componente *Label*, como se ilustra en la Figura 19-17.

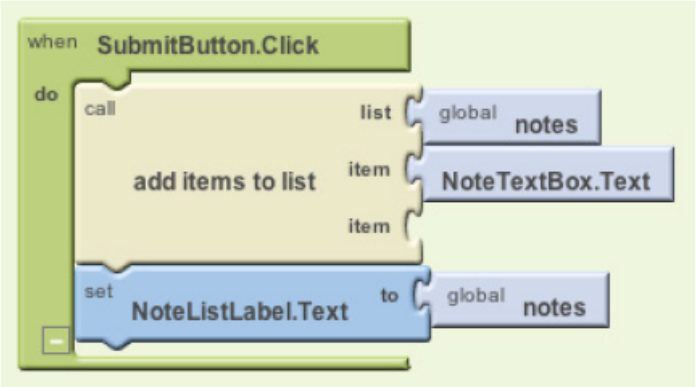

Figure 19-17. Displaying the list to the user within the Text property of the NotesListLabel

Desafortunadamente, este método simple de visualizar una lista no es muy elegante, porque pone la lista dentro de paréntesis con cada artículo separados por un espacio y no necesariamente en la misma línea. Por ejemplo, si el usuario introduce: “Will I ever finish this book? (¿Voy a terminar este libro?)" la primera vez, y: “I forget what my son looks like! (¡me olvido a quien se parece mi hijo!)" la segunda vez, la aplicación le mostrará la lista de notas como se muestra en la figura 19-18.

Si ya ha completado la aplicación "Amazon at the Bookstore" (capítulo 13), este problema le resultará familiar. En el capítulo 20, usted aprenderá cómo mostrar una lista de una manera más sofisticada.

Figure 19-18. The defult list display is not very elegant

Eliminar un elemento “item” de la lista

Puede eliminar un elemento de una lista con el bloque **remove list item**, que se muestra en la Figura 19-19.

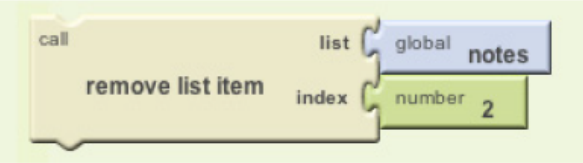

Figure 19-19. Removing an item from a list

Los bloques de la figura 19-19 eliminan el segundo *item* de la lista notes. Sin embargo, generalmente usted no va a querer quitar un elemento fijo (por ejemplo, 2), sino que le permitirá al usuario elegir el elemento que desea eliminar.

ListPicker es un componente de interfaz de usuario que puede ser utilizado para la eliminación de elementos. ListPicker viene con un botón asociado. Cuando se pulsa el botón, ListPicker muestra los elementos de una lista y le permite al usuario elegir uno. Cuando el usuario selecciona un elemento, la aplicación lo puede quitar.

ListPicker es fácil de programar si usted entiende sus dos eventos clave, **BeforePicking** y **AfterPicking** y sus dos propiedades clave, elements y selection, que se enumeran en la Tabla 19-2.

Table 19-2. Two key events of the ListPicker component and their properties

Event	Property
BeforePicking: Triggered when button is clicked.	`Elements`: The list of choices.
AfterPicking: Triggered when user makes a choice.	`Selection`: The user's choice.

Evento	Propiedad
BeforePicking: Disparado cuando se hace clic en el botón.	Elements: La lista de opciones
AfterPicking: Disparado cuando el usuario hace una elección	Selection: La elección del usuario

El evento **ListPicker.BeforePicking** se activa cuando el usuario hace clic en el botón asociado ListPicker, pero antes de enumerar las opciones. En el controlador de eventos **ListPicker.BeforePicking**, podrá establecer la propiedad ListPicker.Elements a una variable de lista. Para la aplicación Note Taker, estableces la propiedad Elements con la variable notes que contiene su lista de notas, como se muestra en la Figura 19-20.

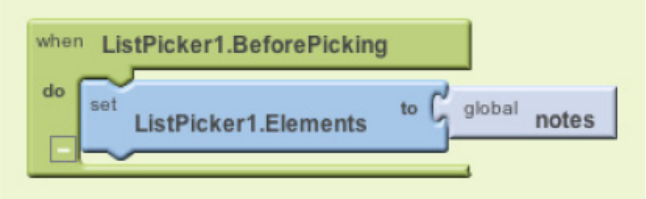

Figure 19-20. The Elements property of ListPicker1 is set to the list contained in notes

Con estos bloques, los elementos de la lista notes aparecerán en la ListPicker. Si había dos notas (elementos), se verá como se muestra en la figura 19-21.

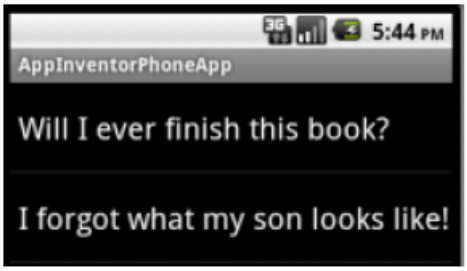

Figure 19-21. The list of notes appears in the ListPicker

Cuando el usuario elige un elemento de la lista, el evento **ListPicker.AfterSelection** se dispara. En este controlador de eventos, se puede acceder a la selección del usuario en la propiedad ListPicker.Selection.

Recuérdese, sin embargo, que el bloque **remove item from list** espera un index (posición de lista), y no un elemento (item). Desafortunadamente, la propiedad Selection de ListPicker son los datos actuales (el item *note*), y no el índice, y el componente ListPicker no proporciona acceso directo al índice de la lista (esto ciertamente será añadido en versiones posteriores de App Inventor).

La solución consiste en tomar ventaja de otro bloque de la Sección Lista, **position in list** (posición en la lista). Dado un texto, esta función devuelve la posición de la primer coincidencia a ese texto en una lista. Usando **position in list**, el controlador de eventos **ListPicker1.AfterPicking** puede eliminar el elemento seleccionado, como lo muestran los

bloques de la figura 19-22

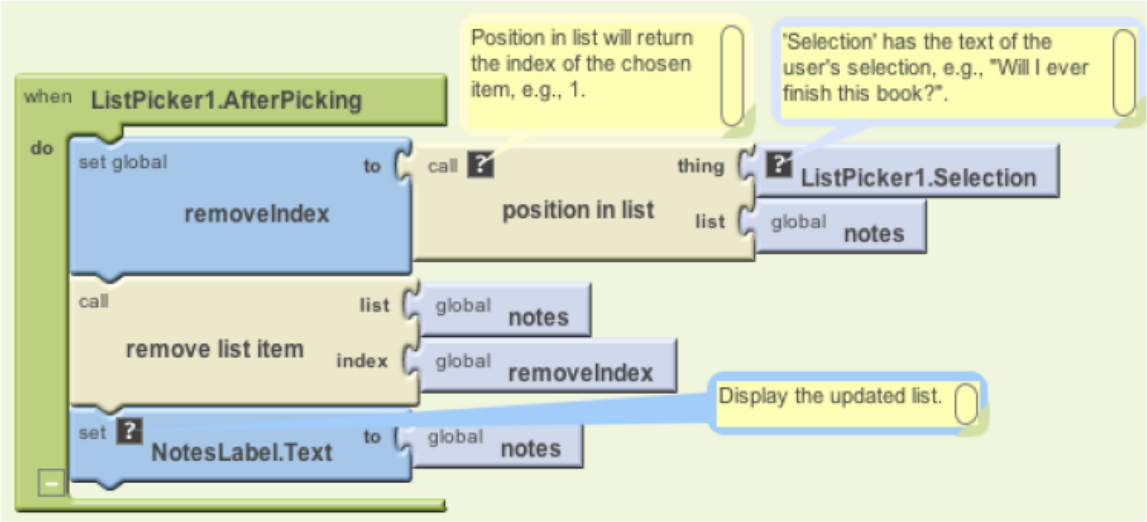

Figure 19-22. Using the position in list block to find the index of the item to remove

Figura 19-22. Utilizando el bloque positioninlist para encontrar el índice del elemento a eliminar
1° nota en la imagen:
Position-in-list devolverá el índice del elemento seleccionado. por ejemplo 1
2° nota en la imagen:
"Selection" tiene el texto que selecciona el usuario, por ejemplo "Will I ever finish this book"
3° nota en la imagen:
Muestra la lista actualizada
.
Cuando **AfterPicking** se dispara, **ListPicker1.Selection** contiene el texto de la elección del usuario (por ejemplo, "Will I ever finish this book?"). El objetivo es encontrar el índice de selección en la lista notes con el fin de eliminarlo, entonces **position in list** es llamado. Si la selección del usuario era "¿Voy a terminar este libro?", **position in list** devuelve 1 porque es el primer elemento. Este número se coloca en la variable removeIndex, que luego será utilizada como índice en la convocatoria a **remove list item**.

He aquí una pregunta para digerir antes de seguir leyendo: ¿crees que este sistema funcionará en todos los casos?

La respuesta es que el sistema funciona bien *a menos que* se dupliquen los datos en la lista. Digamos que el usuario ha escrito: "Estoy teniendo un gran día", tanto en su segunda nota como en la décima. Si hace clic en el botón *remove* de (ListPicker) y elige el décimo punto, el segundo será eliminado en lugar del décimo. **position in list** sólo devuelve el índice del elemento seleccionado y se detiene allí, por lo que nunca se entera de que el décimo punto es el mismo y también debe ser removido de la lista. Habría que incluir algunas comprobaciones condicionales (véase el capítulo 18) para recorrer la lista y ver si había otras entradas que también coincidieron con el elemento seleccionado, y luego eliminar esos también.

Listas de Listas

Los elementos de una lista pueden ser números, texto, colores o valores booleanos (verdadero / falso). Sin embargo, los elementos de una lista también pueden ser listas. Usted verá este tipo de estructuras de datos complejas muy a menudo. Por ejemplo, una lista de listas podría ser utilizado para modificar la aplicación Quiz Presidents del capítulo 8 en una prueba de múltiple-choice (elección múltiple). Vamos a ver de nuevo a la estructura básica de Quiz Presidents, que es una lista de preguntas y una lista de respuestas, como se muestra en la Figura 19-23.

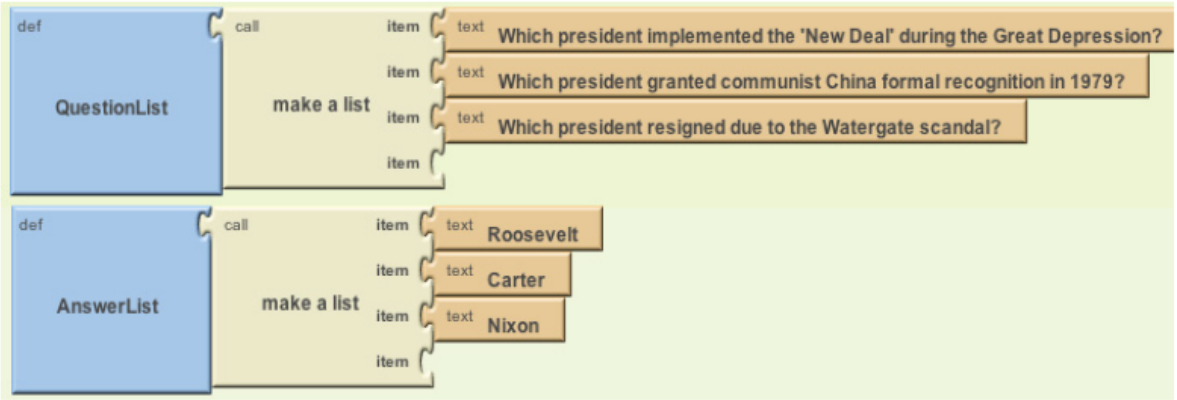

Figure 19-23. A list of questions and a list of answers

Cada vez que el usuario responde a una pregunta, la aplicación comprueba si es correcta comparando la respuesta con el elemento de la AnswerList.

Para realizar la multiple-choice de Quiz, usted necesitaría disponer de una lista de opciones para cada respuesta a cada pregunta. La lista de multiple-choice se conforma con una variable lista de listas, que se define mediante la colocación de tres bloques **make a list** dentro de otro bloque **make a list**, como se muestra en la Figura 19-24.

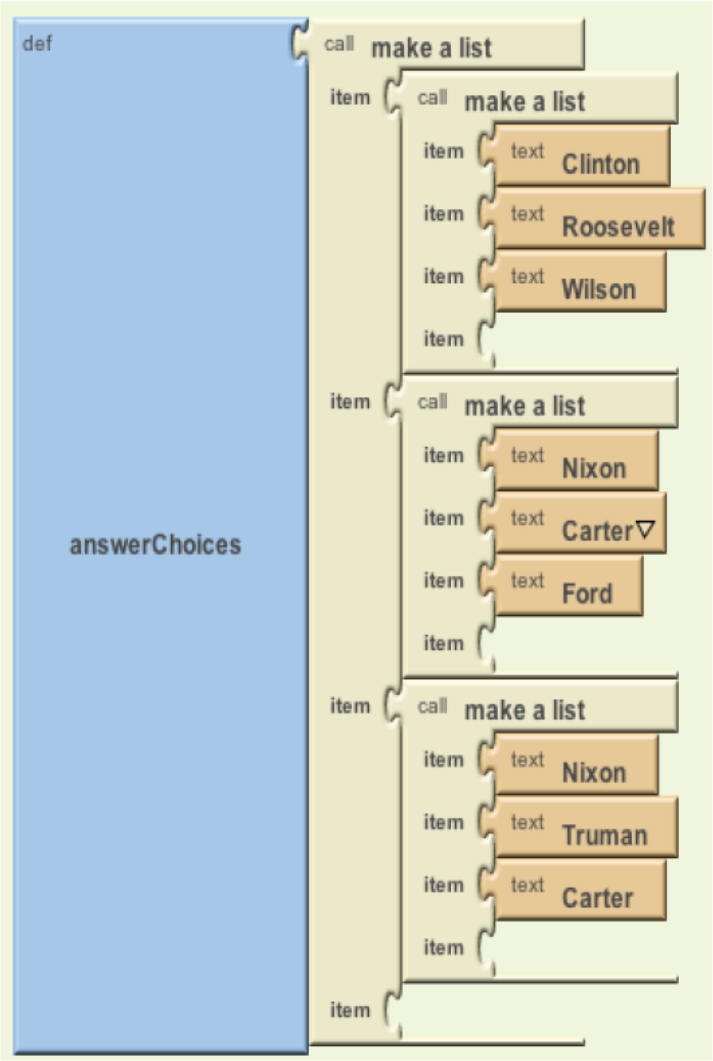

Figure 19-24. A list of lists is formed by inserting make a list blocks as items within an outer make a list block

Cada elemento en la variable answerChoices en sí es una lista que contiene tres artículos. Si selecciona un elemento de answerChoices, el resultado es una lista. Ahora que usted ha rellenado su respuestas de opción múltiple de listas, ¿cómo mostrarlas al usuario?

Al igual que con la aplicación Note Taker, podría utilizar un ListPicker para ofrecerle las opciones al usuario. Si el índice fué nombrado currentQuestionIndex, el evento ListPicker.BeforePicking
aparecerá como se indica en la figura 19-25.

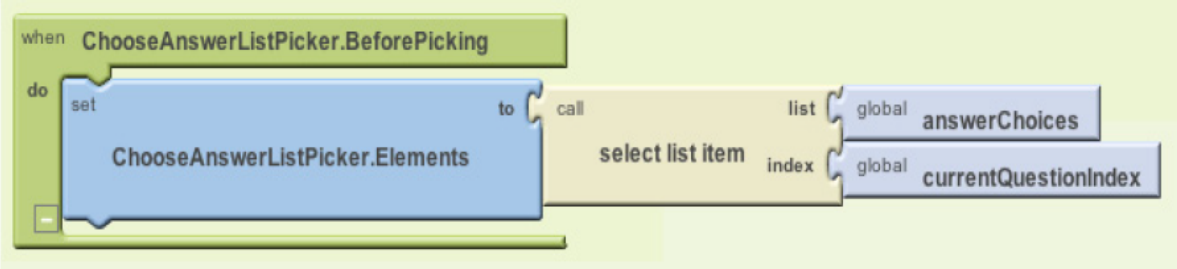

Figure 19-25. Using the List Picker to present the list of choices to the user

Estos bloques toman la sublista actual de answerChoices y permiten al usuario elegir de ella. Entonces, si currentQuestionIndex fuese un 1, el ListPicker mostraría una lista como la de la figura 19-26.

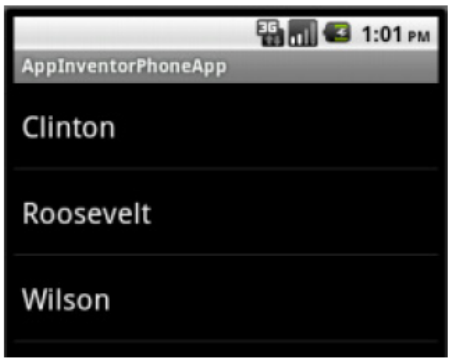

Figure 19-26. The answer choices presented to the user for the second question

Cuando el usuario elige, se comprueba la respuesta con los bloques mostrados en la Figura 19-27.

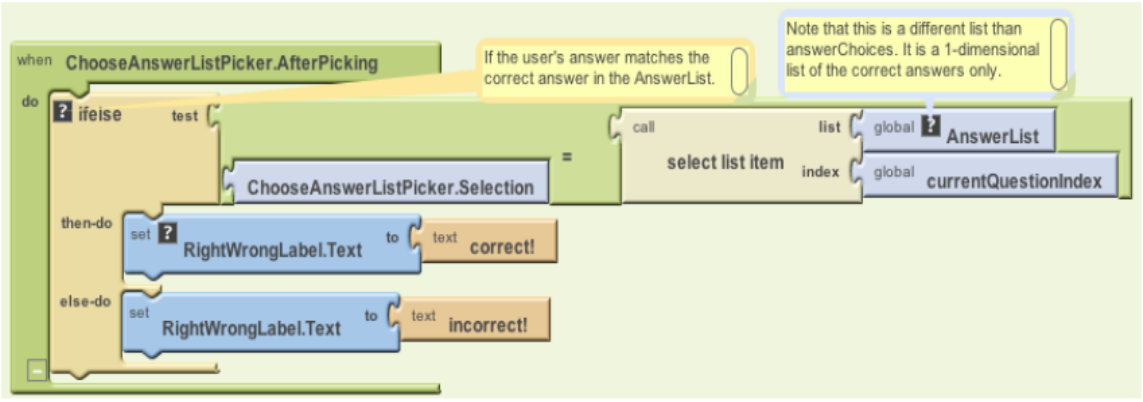

Figure 19-27. Checking whether the user chose the correct answer

En estos bloques, la selección del usuario desde ListPicker se compara con la respuesta correcta, que se almacena en una lista diferente, AnswerList (ya que answerChoices proporciona sólo las opciones y no denota la respuesta correcta).

Resumen

Las listas se utilizan en casi todas las aplicaciones que se le ocurran. Entender cómo funcionan es fundamental para la programación. En este capítulo, se exploró uno de los patrones de programación más comunes: el uso de una variable *index* que comienza en por el principio de la lista y se incrementa hasta que cada elemento de la lista se procesa. Si usted puede entender y adaptar este modelo, es usted un programador!

A continuación vamos a cubrir algunos de los demás mecanismos para la manipulación de listas, incluyendo las formas típicas para permitir al usuario añadir y quitar elementos. Esta programación requiere otro nivel de abstracción, ya que hay que prever los datos, sus listas están vacías hasta que el usuario pone algo en ellas. Si usted entiende esto, puede dejar su tarea por hoy!

Concluimos el capítulo mediante la introducción de una estructura de datos compleja, una lista de listas. Esto es definitivamente un concepto difícil, pero lo hemos explorado utilizando datos fijos: Las opciones de respuestas para un examen de múltiple-choice. Si usted dominó eso y el resto del capítulo, la prueba final es la siguiente: crear una aplicación que utiliza una lista de listas, pero con datos dinámicos! Un ejemplo sería una aplicación que le permite a las personas crear su propio cuestionario múltiple-choice, ampliando aún más la aplicación MakeQuiz en el capítulo 10. ¡Buena suerte!

Mientras piensas como se da solución a esto, entendemos que nuestra exploración de listas no está terminada. En el próximo capítulo, vamos a continuar con el debate y centrarnos en lista de iteración (repetición) con una peculiaridad: la aplicación de las funciones de cada elemento de una lista.

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