SAMSUNG

SAMSUNG

CLUB de APPS

Programa tus Ideas

TUTORIAL DIARIO

DÍA 4

Inria Chile

Introducción

Hola! Bienvenido al cuarto día del taller Programa Tus Ideas :) Tan solo nos van quedando dos días de trabajo !!

Hoy aprenderás a desarrollar aplicaciones *"tipo cuestionario"* o *"trivia"* donde el usuario va avanzando a través de una serie de elementos, que están internamente almacenados en una lista. Desarrollarás como tutorial el juego **4 Fotos 1 Imagen**, donde el usuario debe ingresar la palabra asociada a las 4 fotos que muestra la aplicación. Después verás cómo extender esta y otras aplicaciones. El concepto fundamental que aprenderás haciendo esta aplicación es a iterar sobre una lista utilizando una variable como índice.

También aprenderás a desarrollar aplicaciones que envían mensajes de texto (SMS) y que pueden procesar mensajes de texto recibidos. Desarrollaremos la aplicación **No SMS** al Volante, que responde automáticamente a los mensajes recibidos con una respuesta predeterminada. También aprenderás a manejar una nueva manera de almacenar datos en variables, usando listas. Las listas te permiten tener en un mismo lugar una colección de valores, y realizar operaciones sobre estos valores de manera general, es decir, que no importará si la lista tiene 1, 3 o 100 elementos, tu código será el mismo.

1. Tutorial: 4 Fotos y 1 Palabra

En este tutorial desarrollaremos el juego *4 Fotos y 1 Palabra (4F1P)*, que consiste en mostrar al usuario una serie de *"diapositivas"* que consisten en 4 imágenes, y un campo de texto para que se ingrese la palabra esperada. La interfaz de usuario se muestra en la Figura 1.1.

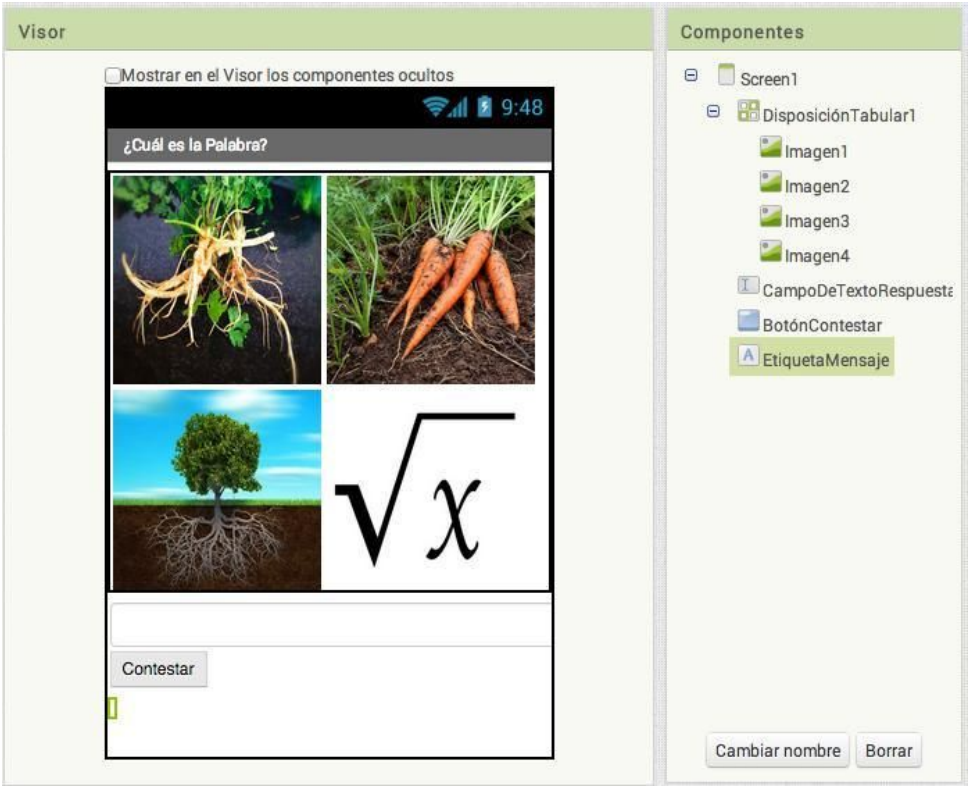

Figura 1.1: Interfaz de usuario 4F1P.

En este tutorial aprenderás a recorrer una lista utilizando una variable como índice, para así implementar una aplicación tipo *"cuestionario"* o *"trivia"*. Este tipo de aplicaciones se encuentran tanto en juegos (como **4F1P**) como en otro tipo de aplicaciones más *"serias"* (Duolingo, Google Forms, etc.).

A continuación desarrollaremos el esqueleto de este juego, que luego podrás completar a tu gusto.

Agregando los Componentes

Para obtener una interfaz de usuario como la de la Figura 1.1 debes utilizar los componentes que se describen en la Tabla 1.1

Tipo de Componente	Nombre	Propósito
DisposiciónTabular	**DisposiciónTabular1**	Aquí pondrás las 4 fotos que se muestran para cada palabra.
Imagen	**Imagen1, Imagen2, Imagen3, Imagen4**	Necesitas 4 componentes de imagen, que van dentro de las celdas de la disposición tabular.
CampoDeTexto	**CampoDeTextoRespuesta**	Aquí el usuario ingresa la palabra asociada a las fotos.
Botón	**BotónContestar**	Al presionar el botón la aplicación averiguar si la respuesta es correcta o no. Si es correcta se pasa a la siguiente palabra.
Etiqueta	**EtiquetaMensaje**	Una etiqueta para mostrarle al usuario un mensaje cuando se equivoca.

Tabla 1.1: Componentes de la interfaz de usuario de **4F1P**.

Una vez que ordenaste los componentes en el Diseñador, tienes que configurarlos de la siguiente manera:

1. Cambia el **Título** de la pantalla a *"¿Cuál es la Palabra?"*.

2. Configura la **DisposiciónTabular1** para que tenga 2 columnas y 2 registros. Ajusta el **Ancho** como “Ajustar al contenedor", y el **Alto** en 300 pixeles.
3. Para las imágenes, ajusta su **Alto** y **Ancho** en 150 pixeles. Cambia la propiedad **Foto** de cada una de ellas con los textos *“raiz1.jpg"*, *“raiz2.jpg"*, *“raiz3.jpg"* y *“raiz4.jpg"*, respectivamente. Los archivos los puedes encontrar en la carpeta **ProgramaTusIdeas/ Dia4**.
4. Cambia la **Pista** del **CampoDeTextoRespuesta1** por el texto “Ingresa la palabra".
5. Borra el texto de la **EtiquetaMensaje** para que está vacío. Este texto lo usaremos cuando el usuario acierte o se equivoque en su respuesta.

Agregando el Comportamiento

El comportamiento de la aplicación debe ser como sigue: cuando el usuario presiona el **BotónContestar**, la aplicación compara el texto ingresado por el usuario, con la palabra esperada. Si la respuesta coincide con la solución, se avanza a la siguiente palabra, y se actualizan las propiedades **Foto** de los componentes de imagen para que muestren la siguiente imagen en la lista.

Implementaremos este comportamiento de la siguiente manera:

1. Definir una variable *listaPalabras* con las palabras que se van a mostrar en la aplicación. Además, definimos una variable *índice* que usaremos para avanzar en la lista. En este tutorial partiremos con 2 palabras, como se muestra en la Figura 1.2.

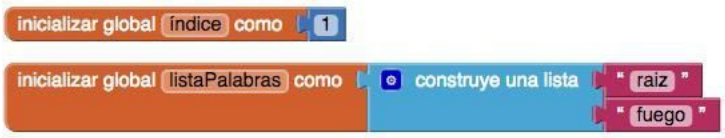

Figura 1.2: Lista de palabras y variable índice.

2. Para cambiar fácilmente las fotos de los componentes de imagen, usaremos la convención de que los archivos serán de la forma “palabra1.jpg", “palabra2.jpg", “palabra3.jpg" y “palabra4.jpg". Esto nos permite hacer un procedimiento que construye los nombres de las fotos y los pone en las propiedades **Foto** de los componentes de imagen. El código se muestra en la Figura 1.3.

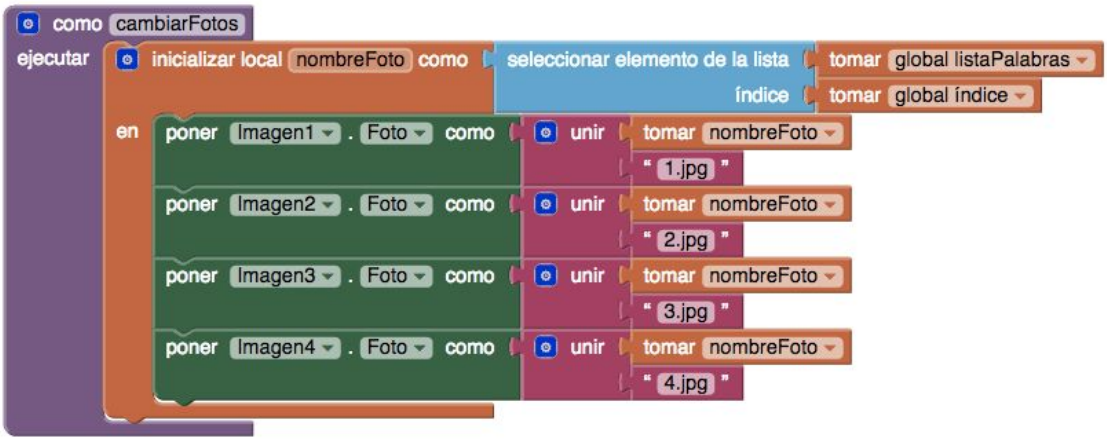

Figura 1.3: Procedimiento para cambiar las imágenes.

Observa que usamos una variable local, ***nombreFoto***, para evitar repetir la selección del elemento de la lista. Esto nos ayuda a disminuir el tamaño del código. Fíjate también en que tomamos la “palabra" desde la *listaPalabras* según el valor actual del índice.

3. Para procesar la respuesta del usuario debemos usar el evento **BotónContestar.Click**. Aquí debemos comparar si el texto ingresado por el usuario coincide con la palabra actual. Si estas , incrementamos el índice y cambiamos las fotos. En caso contrario, mostramos un mensaje al usuario diciéndole que se equivocó . El código se muestra en la Figura 1.4.

Figura 1.4: Verificando la respuesta del usuario.

Para comparar los textos usamos el bloque **comparar textos**, que permite ver si dos textos son iguales, o si uno es “mayor" o “menor" que otro. Para ver cuando un texto es

mayor o menor se considera el orden *lexicográfico* de las letras que lo componen. (En computación, la idea del orden lexicográfico es la misma que la del orden alfabético, pero se usa un nombre distinto porque no necesariamente ambos órdenes coinciden.)

Prueba tu Aplicación! Ahora ya tienes una versión inicial de **4F1P**. Pruébala en tu dispositivo: ¿qué pasa cuando te equivocas? ¿qué pasa cuando respondes correctamente? ¿qué pasa cuando respondes bien la segunda palabra?

Resumen

En este tutorial has desarrollado los siguientes conceptos:

- El componente **EnviarTexto** puede ser utilizado para mandar mensajes de texto y para procesar los mensajes de texto recibidos. Antes de llamar a **EnviarTexto.EnviarMensaje** debes ajustar las propiedades de **NúmeroDeteléfono** y **Mensaje**. Para contestar a un texto recibido, tienes que programar el controlador de eventos **EnviarTexto.MensajeRecibido**.
- El componente **TinyDB** se utiliza para almacenar información de manera persistente en la base de datos del teléfono para que los datos puedan ser cargados cada vez que la aplicación se abre.
- El componente **TextoAVoz** toma cualquier objeto de texto y lo habla en voz alta.
- El bloque **unir** se usa para juntar (concatenar) distintos textos separados en un solo objeto de texto.

2. Tutorial: No SMS al Volante

Contexto En Enero de 2010, el Consejo Nacional para la Seguridad en Estados Unidos anunció los resultados de un estudio que descubrió que al menos 28 % de los accidentes de auto eran causados por personas utilizando teléfono celular mientras estaban manejando.

Daniel Finnegan, un estudiante del curso de programación AppInventor en la Universidad de San Francisco, tuvo una muy buena idea para ayudar con la epidemia de mandar textos mientras se conduce. La aplicación que creo, mostrada en la Figura 2.1, contesta

automáticamente (sin tener que usar el teléfono) a cualquier mensaje de texto recibido, con un mensaje del estilo: “Estoy manejando, te contactare a la brevedad".

Discusiones durante la clase lo llevaron a crear funcionalidades adicionales:

- El usuario puede cambiar la respuesta automática dependiendo de la situación (por ejemplo si estás en el cine, si estas en una reunión, etc.).
- La aplicación puede hablar el texto recibido en voz alta (aunque sabes que la aplicación va a mandar una respuesta automática, puedes estar muy curioso por saber que mensaje te mandaron).

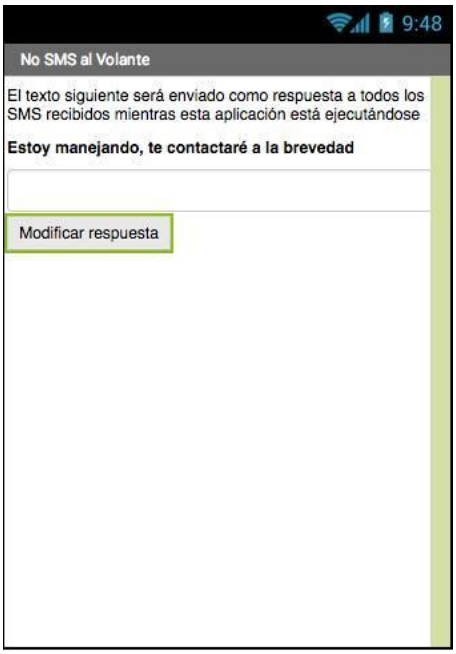

Figura 2.1: La aplicación **No SMS al Volante**.

Qué Aprenderás

Es una aplicación más compleja que las de las actividades previas. Entonces, la vas a construir paso por paso, una funcionalidad a la vez, empezando con el mensaje de respuesta automática. Aprenderás sobre:

- El componente **EnviarTexto** para enviar mensajes de textos y procesar los mensajes recibidos.
- Un formulario de entrada para someter el mensaje de respuesta personalizada.
- El componente de base de datos **TinyDB** para guardar o preservar el mensaje personalizado después del cierre de la aplicación.

- El evento **Screen.Inicializar** para cargar la respuesta personalizada cuando la aplicación se comienza a ejecutar.
- El componente **TextoAVoz** para escuchar el texto de los mensajes recibidos en voz alta.

Diseñar los componentes

La interfaz de usuario para esta aplicación es bastante simple: tiene una etiqueta que muestra la respuesta automática, con una caja de texto y un botón para agregar un cambio.

También tendrás que seleccionar un componente **EnviarTexto**, un componente **TinyDB**, y un componente TextoAVoz, y un componente **SensorDeUbicación**.Todos aparecerán en la zona de componentes no visibles. Los componentes de tu proyecto debieran verse como en la Figura 2.2.

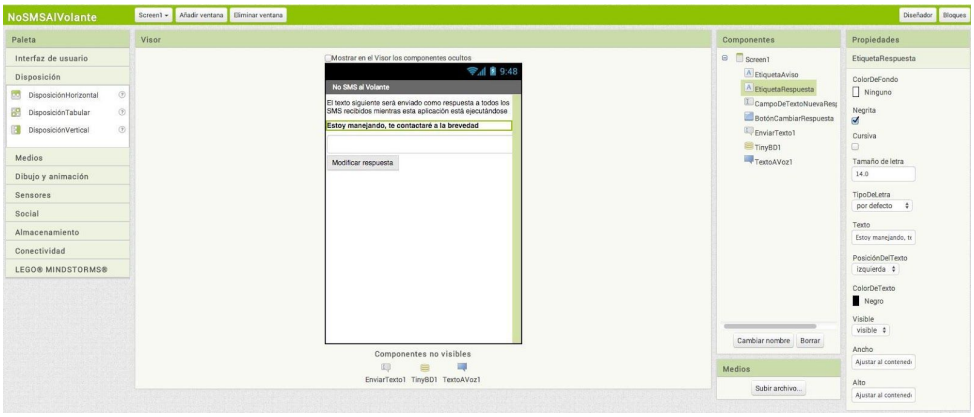

Figura 2.2: La aplicación **No SMS al Volante** en el Diseñador de Componentes.

Puedes construir la interfaz de usuario mostrada en la Figura 2.2 seleccionando los componentes listados en la Tabla 2.1.

Luego debes con gurar los componentes de la siguiente manera:

- **Texto** de **EtiquetaAviso**: “El texto siguiente será enviado como respuesta a todos los SMS recibidos mientras esta aplicación está ejecutándose".
- **Texto** de **EtiquetaRespuesta**: “Estoy manejando, te contactare a la brevedad". Selecciona la propiedad Negrita.
- **Texto** de **CampoDeTextoNuevaRespuesta**: (Deja el campo de texto en blanco para que el usuario pueda escribir en el).

Tipo Componente	Nombre	Propósito
Etiqueta	EtiquetaAviso	Dice al usuario como funciona la aplicación
Etiqueta	EtiquetaRespuesta	La respuesta que se enviar al remitente del mensaje de texto recibido
CampoDeTexto	CampoDeTextoNuevaRespuesta	El usuario ingresará una respuesta personalizada aquí
Botón	BotónCambiarRespuesta	El usuario presiona el botón para cambiar la respuesta
EnviarTexto	EnviarTexto1	Procesa los mensajes de texto
TinyDB	TinyDB1	Almacena la respuesta en una base de datos
TextoAVoz	TextoAVoz1	Lee en voz alta los mensajes recibidos

Tabla 2.1: Todos los componentes de la aplicación No SMS al Volante.

- La **Pista** de **CampoDeTextoNuevaRespuesta**: “Escribir texto de nueva respuesta".
- **Texto** de **BotonCambiarRespuesta**: “Modificar respuesta".

Agregar Comportamiento a los Componentes

Empezarás por programar el comportamiento de la respuesta automática con texto básico, y luego añadir las funcionalidades de manera sucesiva.

Programar una respuesta automática

Para el comportamiento de la respuesta automática, utilizaras el componente **Enviar Texto de AppInventor**. Puedes imaginar que este componente es una pequeña persona dentro de tu teléfono que sabe leer y escribir texto. Para leer textos, el componente usa el block de evento **EnviarTexto.MensajeRecibido**. Puedes seleccionar este bloque y color otros bloques adentro para ver lo que pasar a cuando un mensaje de texto es recibido. En el caso de la aplicación que vamos a programar, queremos enviar de manera automática una respuesta de texto escrito anteriormente. Para programa el texto de la respuesta, colocarás un bloque **EnviarTexto1.EnviarMensaje** dentro del controlador de eventos **EnviarTexto1.MensajeRecibido**.

Bloque	Propósito
EnviarTexto1.MensajeRecibido	El controlador de evento que se gatilla cuando el teléfono recibe un mensaje de texto.
poner EnviarTexto1.NúmeroDeteléfono	Con gurar el Número de teléfono al cual se enviar el mensaje.
tomar Número	El Número de teléfono de la persona que envio el mensaje recibido.
poner EnviarTexto1.Mensaje	Con gurar el mensaje que será enviado.
EtiquetaRespuesta.Texto	El mensaje que el usuario con gur para ser enviado.
EnviarTexto1.EnviarMensaje	Enviar el mensaje.

Tabla 2.2: Bloques para enviar una respuesta automática.

El bloque **EnviarTexto.EnviarMensaje** envía el texto, pero debes con anterioridad especificar qué texto hay que enviar y a quién hay que enviarlo, antes de utilizar este bloque. La Tabla 2.2 muestra todos los bloques que necesitarás para programar el comportamiento de respuesta automática, y la Figura 2.3 muestra cómo deben conectarse en el Editor de Bloques.

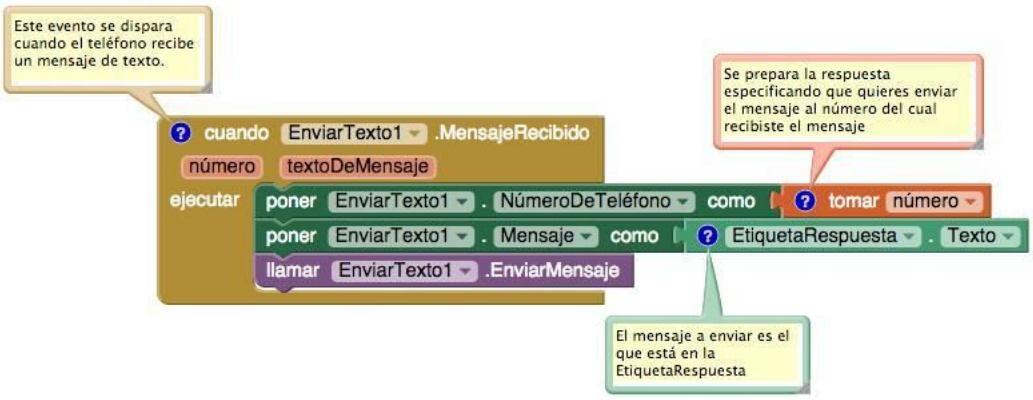

Figura 2.3: Contestar a un mensaje de texto entrante

Cuando el teléfono recibe un mensaje de texto, el evento **EnviarTexto1.MensajeRecibido** se activa. Como lo muestra la Figura 2.3, el Número de teléfono del que llama (el remitente) se encuentra en el argumento Número, y el mensaje de texto recibido está en el argumento *textoDeMensaje*.

Para la respuesta automática, la aplicación necesita mandar un mensaje de texto al remitente. Para enviar el texto, tienes que configurar dos propiedades del componente **EnviarTexto1**: el Número de teléfono del destinatario, **EnviarTexto1**.**NúmeroDeteléfono**, y el mensaje a enviar, **EnviarTexto1.Mensaje**.

Para hacer esto, asigna remitente. Para enviar el texto, tienes que configurar dos propiedades del componente **EnviarTexto1**: el Número de teléfono del destinatario, **EnviarTexto1.NúmeroDeteléfono**, y el mensaje a enviar, **EnviarTexto1.Mensaje**. Para hacer esto, asigna el valor del Número de teléfono del remitente en NúmeroDeteléfono y el texto que escribiste en **EtiquetaRespuesta** (“Estoy manejando, te contactare a la brevedad") en **EnviarTexto1.Mensaje**. Una vez que estas propiedades están configuradas, la aplicación llama a **EnviarTexto1.EnviarMensaje** para enviar la respuesta. Los comentarios constituyen una parte sumamente importante en la programación. Otros programadores pueden usarlos para informarse sobre el código. Puedes añadir comentarios con un click derecho sobre un bloque y seleccionando ”Añadir Comentario". Una vez que has añadido comentarios, puedes mostrarlos o esconderlos haciendo un click en el signo de interrogación azul que aparece. No es

obligatorio que incluyas comentarios en tu aplicación, los incluimos en el ejemplo para ayudar a describir cada bloque y lo que hace.

La mayor a de las personas utilizan comentarios para documentar cómo están construyendo su aplicación; los comentarios explican cómo el programa funciona, pero no modifican la aplicación ni la hacen comportarse de manera distinta. Los comentarios son importantes, tanto para ti (estás construyendo tu aplicación y seguramente la modificaras en el futuro), como para otras personas que podrán querer personalizarla. Todo el mundo está de acuerdo para decir que un software cambia y se transforma a menudo. Por esta razón, comentar el código es muy importantes en ingeniería de software, y especialmente con software libre (de código abierto o open source) como AppInventor.

¡Prueba tu Aplicación! Necesitarás un segundo teléfono para probar la aplicación. Pídele a tus compañeros que te manden un mensaje de texto mientras tienes la aplicación ejecutándose. El teléfono de tu compañero debe recibir la respuesta automática.

Escribir una respuesta personalizada

En esta segunda etapa, vamos a añadir bloques para que el usuario pueda escribir su propia respuesta personalizada. En el Diseñador de Componentes añadiste un componente **CampoDeTextoNuevaRespuesta** que es donde el usuario podrá entrar su respuesta personalizada. Cuando el usuario presiona el botón **BotonCambiarRespuesta** tienes que copiar el texto ingresado en **CampoDeTextoNuevaRespuesta** hacia la etiqueta **EtiquetaRespuesta**, cuyo texto se usa para contestar automáticamente los mensajes de texto.

La Tabla 2.3 muestra los bloques que necesitarás para transferir una nueva respuesta personalizada hacia la **EtiquetaRespuesta**.

Piensa en el funcionamiento de un formulario típico: primero entras un texto a un *campo de texto*, luego presionas un botón para decir al sistema que procese el texto. El formulario de esta aplicación no es distinto para nada!

Bloque	Propósito
BotonCambiarRespuesta.Click	El usuario presiona el botón para configurar la nueva respuesta.
poner EtiquetaRespuesta.Texto	Cambia el texto de la etiqueta por la nueva respuesta.
CampoDeTextoNuevaRespuesta	El usuario ingresa aquí el texto de la nueva respuesta.

Tabla 2.3: Bloques para respuesta personalizada.

La Figura 2.4 muestra cómo utilizar los bloques para procesar la respuesta al evento **BotonCambiarRespuesta.Click** cuando el usuario presiona dicho boton.

Figura 2.4: Cambiar la respuesta cuando el usuario entra un mensaje personalizado.

En este caso, el controlador de evento copia lo que el usuario escribió en **CampoDeTextoNuevaRespuesta** hacia la **EtiquetaRespuesta**. Recuerda que la **EtiquetaRespuesta** corresponde al mensaje que va a ser enviado en la respuesta automática, entonces quieres estás seguro que este mensaje es el nuevo mensaje personalizado que el usuario acaba de entrar.

¡Prueba tu Aplicación! Ingresa una respuesta personalizada y valídala. Luego pide a un compañero que te envíe un mensaje de texto mientras ejecutas la aplicación. ¿Recibió tu compañero la respuesta correcta?

Almacenar la Respuesta Personalizada en una Base de Datos

Ya tienes una excelente aplicación, pero con un defecto: si el usuario entra una respuesta personalizada, cierra la aplicacion y despues la ejecuta de nuevo, la nueva respuesta

automática no aparecer . Este no es el comportamiento que el usuario esperar a; quieren ver la respuesta personalizada cuando lanzan la aplicación de nuevo. Para que esto pase, tienes que almacenar persistentemente la respuesta personalizada.

Bloque	Propósito
TinyDB1.GuardarValor	Almacena la respuesta personalizada en la base de datos del teléfono.
Bloque de texto **"mensajeRespuesta"**	Nombre o etiqueta para identificar el mensaje de respuesta en la base de datos.
EtiquetaRespuesta.Texto	Aquí se muestra el mensaje de respuesta.

Tabla 2.4: Bloques para almacenar la respuesta personalizada con TinyDB.

Podrás pensar que poner datos en la propiedad **EtiquetaRespuesta.Texto** corresponde técnicamente a almacenarlos, pero el problema es que estos datos almacenados en propiedades de componentes son datos transitorios. Los datos transitorios son como tu memoria a corto plazo; el teléfono los olvida apenas cierra la aplicación. Si quieres que la aplicación recuerde algo de manera persistente, tienes que transferir los datos desde la memoria a corto plazo (una propiedad de componente o una variable) a la memoria a largo plazo (una base de datos).

Para almacenar datos de manera persistente, usaras el componente TinyDB que guarda datos en una base de datos que se encuentra en el dispositivo Android. TinyDB provee dos funciones: **GuardarValor** y **ObtenerValor**. La primera permite a la aplicación almacenar información en la base de datos del dispositivo, mientras la segunda permite a la aplicación acceder a datos que están ya almacenados.

Para varias aplicaciones, usarás el esquema siguiente:

1. Almacenar datos en la base de datos cada vez que el usuario entra un valor nuevo.
2. Cuando la aplicación se inicia, cargar las datos desde la base de datos y guardarlos en variables o propiedades.

Empezaras por modificar el controlador de evento **BotonCambiarRespuesta.Click** para que almacene datos de manera persistente, usando los bloques listados en la Tabla 2.4.

Esta aplicación utiliza **TinyDB** para tomar el texto que está en **EtiquetaRespuesta** y almacenarlo en la base de datos. Como lo muestra la Figura 2.5, cuando almacenas datos en la base de datos, tienes que marcar o etiquetar tus datos, para poder recuperarlos posteriormente. En este caso la etiqueta es *"mensajeRespuesta"*.

Bloque	Propósito
Definir variable global *respuesta*	Una variable temporal para almacenar los datos obtenidos
Bloque de texto vacío	El valor inicial para la variable, puede ser cualquier cosa
TinyDB1.ObtenerValor	Obtiene el valor almacenado en la base de datos
Bloque de texto **"mensajeRespuesta"**	Argumento para el parámetro etiqueta de la función **TinyDB1.ObtenerValor.** Asegúra te que la etiqueta es la misma que la usada para guardar los datos anteriormente.
Bloque **longitud** (sección *"Texto"*)	Para chequear si el texto obtenido de la base de datos tiene largo mayor que 0 (o sea, si está vacío o no).

Tabla 2.5: Bloques para cargar los datos de vuelta al abrir la aplicación

Figura 2.5: Almacenando la respuesta personalizada de manera persistente.

Recuperar la Respuesta Personalizada al Abrir la Aplicación

La razón por la cual queremos almacenar la respuesta personalizada en la base de datos es para poder recuperar la respuesta de vuelta la próxima vez que el usuario abre la aplicación. Como ya hemos mencionado anteriormente, AppInventor provee un bloque de evento que se activa cuando la aplicación se abre: **evento especial que está activado cuando la aplicación se abre:** Screen1.Inicializar. Si seleccionas este bloque y colocas otros bloques adentro, estos serán ejecutados justo al abrir la aplicación.

Para esta aplicación, el controlador de evento **Screen1.Inicializar** debe averiguar si una respuesta personalizada está almacenada en la base de datos. De ser así, la respuesta personalizada deber a ser cargada usando la función **TinyDB.ObtenerValor**. Los bloques principales que necesitarás se muestran en la Tabla 2.5.

La Figura 2.6 muestra el código necesario. Para entenderlo, tienes que visualizar a un usuario abriendo la aplicación por primera vez, ingresando una respuesta personalizada, y abriendo la aplicación varias veces después. La primera vez que el usuario abre la aplicación, no haber ninguna respuesta personalizada en la base de datos a cargar, entonces tienes que poner la respuesta prescrita (respuesta por defecto) en **EtiquetaRespuesta**. Al ejecutar la aplicación las próximas veces, cargaras las respuestas personalizadas almacenadas desde la base de datos y las colocaras en **EtiquetaRespuesta**.

Figura 2.6: Cargando la respuesta personalizada desde la base de datos en la inicialización de la aplicación.

Cuando la aplicación empieza, el evento **Screen1.Inicializar** se activa. La aplicación llama a **TinyDB1.ObtenerValor** con una etiqueta *"mensajeRespuesta"* la misma etiqueta que utilizaste cuando almacenaste la respuesta personalizada del usuario un tiempo atrás. El valor correspondiente se coloca en la variable respuesta para que puedas inspeccionarla antes de poner el valor en **EtiquetaRespuesta**.

Pregunta: ¿por qué razón quieres poder inspeccionar el valor obtenido de la base de datos antes de mostrarlo como respuesta personalizada al usuario?

Respuesta: TinyDB retorna un texto vacío si no hay datos correspondiente a la etiqueta especificada como argumento en la base de datos. No habrá datos la primera vez que la aplicación se abre; y seguirá siendo las hasta que el usuario entre una respuesta personalizada.

La variable *response* contiene el valor almacenado en la base de datos, entonces podemos usar un bloque condicional para averiguar si el largo de lo que fue retornado desde la base de datos es mayor a 0. Si es as , la aplicación sabe que **TinyDB** retorn algo, y el valor correspondiente puede ser colocado en **EtiquetaRespuesta.** Si el largo no es superior a 0, la aplicación sabe que no hay una respuesta personalizada que haya sido almacenada, entonces no modifica la **EtiquetaRespuesta**, dejando as la respuesta por defecto.

¡Prueba tu Aplicación! No puedes testear este comportamiento en vivo, por que la base de datos de **TinyDB** se borra cada vez haces *"Connect AI Companion"* para recargar la aplicación.

Lo que debes hacer es seleccionar *"Generar"* y luego *"App (generar código QR para el archivo .apk)"*; después escanea el codigo de barra, y luego descargas la aplicación en tu teléfono. Una vez que la aplicación estś instalada, ingresa una nueva respuesta personalizada. Después cierra la aplicacion y vuelve a abrirla. ¿Aparece el mensaje personalizado que pusiste?

Leer en Voz Alta el Mensaje de Texto Recibido

Ahora modificarás la aplicación de tal manera que cuando recibes un mensaje de texto, se habla en voz alta el Número de teléfono del remitente y su mensaje de texto. La idea es que cuando estas manejando y escuchas un mensaje de texto llegar, va a estar tentado de leer el texto incluso si sabes que la aplicación va a mandar una respuesta automática. Con la funcionalidad de **texto-a-voz**, puedes escuchar el texto y seguir manejando.

Los dispositivos Android vienen con capacidades de **texto-a-voz** y AppInventor provee un componente, **TextoAVoz**, que hablar cualquier texto que le des (Nota: por "texto" se entiende una secuencia de letras, cifras y puntuación, no una mensaje de texto obligatoriamente). El componente **TextoAVoz** es muy simple de usar: solamente llamas su función **Hablar** y le das como argumento el texto que quieres que el teléfono pronuncie. Por ejemplo, la función mostrada en la Figura 2.7 diría *"Hola Mundo".*

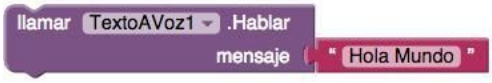

Figura 2.7: Bloques para decir “Hola Mundo" en voz alta.

Para la aplicación **No SMS al Volante**, necesitarás hacer hablar un mensaje más complicado, que contiene el texto recibido y el mensaje de la persona que lo mando. En vez de conectar un objeto estático como el bloque de texto *“Hola Mundo"*, conectarás un bloque **unir** (de la sección “Texto"). El bloque **unir** te permite combinar piezas separadas de texto (o Números u otros caracteres) en un solo objeto de texto.

Bloque	Propósito
TextoAVoz.Hablar	Pronuncia en voz alta el mensaje de texto recibido.
Bloque **unir**	Construye el texto completo que será leído en voz alta.
Bloque de texto “**Mensaje recibido de**"	Las primeras palabras que se dirán.
tomar NúmeroDeteléfono	El Número del remitente del mensaje recibido.
tomar mensajeDeTexto	El mensaje recibido.

Tabla 2.6: Bloques para hablar el texto entrante en voz alta.

Necesitarás llamar a **TextoAVoz.Hablar** dentro del controlador de eventos **EnviarTexto.MensajeRecibido** que programaste anteriormente. Los bloques que programaste hace algunos pasos atrás manejan el evento ajustando las propiedades NúmeroDeteléfono y Mensaje, y luego enviando la respuesta de texto. Debes extender este controlador de evento añadiendo los bloques de la Tabla 2.6.

Después de enviar la respuesta automática, la aplicación llama a la función **TextoAVoz.Hablar**, como lo muestra la Figura 2.8. Puedes conectar cualquier objeto de texto en el parámetro mensaje. En este caso,se usa **unir** para construir las palabras por ser habladas.

Concatena (o añade, o pega juntos) el texto *“Mensaje recibido de "* y el Número de teléfono del remitente, seguido por el texto. *“El mensaje es"*, y finalmente el mensaje recibido. Entonces, si el texto *“Hola"* fue enviado desde el Número *“111-2222"*, el teléfono dirá: *“Mensaje recibido de 111-2222. El mensaje es Hola"*.

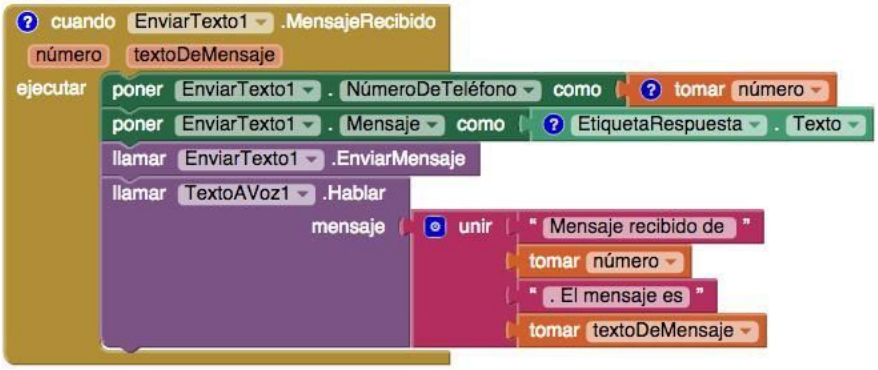

Figura 2.8: Hablando en voz alta el texto recibido.

¡Prueba tu Aplicación! Necesitarás que alguien te envíe un mensaje de texto mientras ejecutas la aplicación. ¿Lee en voz alta el mensaje tu teléfono? ¿Recibe el otro teléfono la respuesta automática correcta?

3. Material de Apoyo

Listas

Las listas son un tipo de objeto que permite almacenar varios valores al mismo tiempo. También la idea de las listas es poder ejecutar tareas repetitivas sobre cada uno de los elementos de una lista. Por ejemplo, podemos usar una lista para enviar un mensaje de texto a 3 compañeros.

Solución sin Usar Listas

Consideremos una aplicación con un botón **BotonEnviarMensajes**. La Figura 3.1 muestra como enviar 3 mensajes de texto usando el componente **EnviarTexto**.

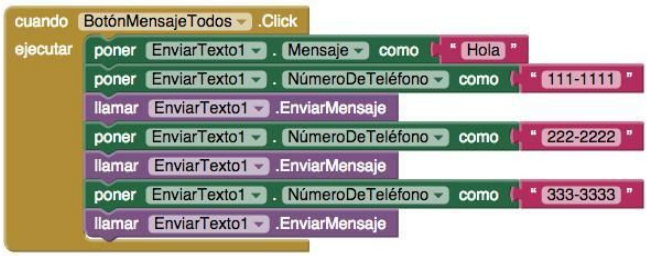

Figura 3.1: Enviar mensaje de texto a 3 Números de teléfono, sin usar listas.

Como puedes ver, la solución simplemente consiste en repetir 3 veces la configuración y ejecución del bloque **EnviarTexto.EnviarMensaje**. ¿Qué pasa ahora si queremos enviar el mensaje a 4 amigos? ¿Y si son 10? ¿Y si son 100? Ya no es muy entretenido agregar tantos bloques, y además hay mucha posibilidad de error.

Solución Usando Listas

Utilizando listas podemos tener una parte del código que es general, sin importar la cantidad de destinatarios de nuestro mensaje, y una parte particular, que es la lista de los Números de teléfono. Para esta solución debes seguir los siguientes pasos:

- Definir una variable global para guardar los Números de teléfono.
- conectar un bloque **construye una lista**, y presionar el cono azul para darle 3 campos.
- Conectar 3 bloques de texto a la lista, que serán los 3 Números de teléfono.
- Utilizar el bloque **por cada elemento en la lista**. Este bloque va a *recorrer* o *iterar* por cada uno de los elementos de la lista, y para cada uno de ellos ejecutar el contenido dentro de su parámetro *"ejecutar"*.
- Por lo tanto, si hay 3 elementos en la lista, los bloques dentro del **por cada** ... serán ejecutados 3 veces. En la ejecución de estos bloques, el parámetro elemento hace referencia al elemento de la lista que está siendo procesado.

El código de la aplicación usando listas se muestra en la Figura 3.2.

Figura 3.2: Enviar mensaje de texto a 3 Números de teléfono, usando listas.

Las listas de datos se encuentran en muchas aplicaciones! Por ejemplo, cuando usas Facebook tienes una lista de tus amigos, una lista de tus mensajes, etc. En tus aplicaciones, podrás querer monitorear los Números de teléfono de tus amigos, una lista de tus puntos en un juego, o el Número de kilómetros que corriste cada día de la semana pasada.

Dado que en casi todos los sistemas de software hay listas de datos involucradas AppInventor, como la mayor a de los lenguajes de programación, provee funcionalidades para procesar los elementos de una lista y realizar la misma operación en cada elemento. Con AppInventor usas un bloque **por cada elemento**... Veamos otro ejemplo.

Ejemplo: ¿Cómo sumar una lista de Números? En el ejemplo anterior, la lista de Números de teléfono está fija. Pero en general, las aplicaciones trabajan con listas dinámicas generadas con datos ingresados por el usuario. Por ejemplo, en una aplicación deportiva el usuario ingresará el Número de kilómetros que corre cada día. Consideremos como ejemplo de esto la lista de la Figura 3.3, que muestra cómo sumar los kilómetros recorridos cada día.

Figura 3.3: Aplicación que suma los kilómetros diarios recorridos por el usuario.

En este ejemplo podemos observar lo siguiente:

- El bloque **por cada** se usa para iterar sobre los Números de la lista.
- La variable total se usa para calcular el resultado total de la suma. Se inicializa con su valor en 0.
- El parámetro elemento primero tiene valor 7, luego 3, luego 11, y finalmente 5.

- En la primera iteración, total tiene su valor previo (0) más el valor de elemento (7), entonces se ajusta el total a 0 + 7 = 7.
- En la segunda iteración, total tiene su valor previo (7) más el valor de elemento (3), entonces se ajusta el total a 7 + 3 = 10.
- Después de la tercera iteración total vale 21, y después de la cuarta iteración total vale 26, y se termina la ejecución del bloque **por cada**.
- Luego de la iteración, se escribe el total en una etiqueta para mostrarlo al usuario.

Datos Persistentes

Cuando ingresas información en tu página de Facebook o en otras aplicaciones, tu esperas que esta información esté guardada para que la próxima vez que visites la página, lo que pusiste siga ah ! Lo mismo pasa con AppInventor: a medida que trabajas en tus proyectos, puedes cerrar la página para luego volver y continuar con el trabajo en el lugar en que te quedaste. Cuando la información se guarda y aparece la próxima vez que visitas una página o abres una aplicación, se dice que los datos son persistentes. En contraste, los datos que solo duran mientras la página o aplicación están abiertos se conoce como datos transitorios. Puedes pensarlo también como la memoria a largo plazo y la memoria a corto plazo de las aplicaciones.

En AppInventor, la memoria a corto plazo o transitoria se expresa en las propiedades de los componentes y en las variables que uses en tus aplicaciones. Para los datos persistentes debes usar el componente **TinyDB**, que provee dos funciones esenciales:

- **TinyDB.GuardarValor** que recibe una etiqueta que identifica los datos almacenados, y el valor que quieres almacenar en la base de datos.
- **ObtenerValor**, que busca en la base de datos si hay información asociada a una etiqueta específica. Si hay datos, los devuelve, y si no los hay te entrega un valor por defecto que tu debes configurar.

Cuando creas aplicaciones con datos persistentes debes preocuparte primero de guardar los datos! Pero además, debes considerar lo siguiente:

- ¿Cuando es necesario recuperar los valores desde la base de datos? En general, es buena práctica recuperarlos justo antes de que sean necesarios. En el tutorial era necesario cargar los datos cuando la aplicación se inicializa, pero no necesariamente debe ser siempre as .
- ¿Qué pasa si no hay datos almacenados? Tu aplicación debe considerar el uso de valores por defecto mientras no haya información almacenada en la base de datos.

Preguntas y Discusión

Considera las soluciones siguientes para mandar un mensaje de texto a una serie de Números de teléfono, que se mostraron anteriormente en la Figura 3.1 y la Figura 3.2.

1. Si tienes que añadir un elemento a la lista de Números de teléfono, ¿para que solución sería más fácil realizar el cambio?
2. ¿Cuántas funciones (llamadas y ajustes de configuración) se ejecutan en la primera solución? Sin contar el bloque por cada y el controlador de evento.
3. ¿Cuántas funciones se ejecutarían si en la lista hubiera n elementos?
4. El código de la primera solución podría ser más eficiente, es decir, que podrán ejecutarse menos bloques si cambiaras algunas cosas. ¿Que bloques moverías y por qué?
5. Las aplicaciones anteriores están diseñadas para un usuario en específico. Es decir, una sola lista de teléfonos para un usuario del teléfono. ¿Si quisieras que la aplicación fuera más general, como la cambiarías?
6. Si quieres hacer que la lista de Números de teléfono sea dinámica, es decir que quieres permitir al usuario ingresar y cambiar los Números de teléfono que van a recibir un mensaje, cuál solución sería mejor? ¿Por qué?

Ejercicios de Personalización

¡Te invitamos a realizar los siguientes desafíos de programación! También puedes personalizar la aplicación a tu gusto!

1. Escribe una versión de **No SMS al Volante** que deja al usuario definir respuestas personalizadas para ciertos Números de teléfono. Por ejemplo: tu mejor amigo te manda un mensaje de texto, contestas con un mensaje, pero para cualquier otra persona contestas con otro mensaje. Necesitarás añadir bloques condicionales para distinguir los Números de teléfono de los remitentes.
2. Escribe una versión de la aplicación que emite una una alarma cuando un mensaje de texto es recibido desde un Número que se encuentra dentro de una lista especial de notificación.
3. Escribe una aplicación que manda mensajes de texto a todos los usuarios de una lista, pero permite al usuario entrar y cambiar los Números de la lista y cambiar el mensaje a enviar. Los datos deben ser persistentes.
4. Escribe una aplicación que guarda notas escritas por el usuario. Las notas deben poder editarse, y los datos deben ser persistentes. ¿Cómo implementarías que las notas puedan ser borradas?

4. Desafíos de Programación

En el tutorial desarrollaste una versión preliminar del juego **4F1P**. Sin embargo, la aplicación presenta el problema de que se *"cae"* cuando respondes correctamente la última pregunta. El error ocurre porque la lista de palabras tiene 2 elementos, y el código intenta obtener el tercero ¡que no existe!

A la aplicación también le faltan varias cosas para ser un buen juego: un marcador de puntaje, quizás puedas tener una cantidad de vidas, un botón para dar pistas, etc.

Antes de mejorar esta aplicación te proponemos que revises algunos conceptos y desafíos en una aplicación de ejemplo que es un poco más sencilla. La aplicación se llama **Slideshow** y te permite avanzar por una lista de *"diapositivas"*.

Personaliza y Crea Tus Propias Aplicaciones

Puedes extender y mejorar las aplicaciones **Slideshow** y **4F1P**. Además, te proponemos que desarrolles otra aplicación Identifica a tus Compañeros. En esta aplicación puedes mostrar la foto de tus compañeros y debes escribir su nombre. Te damos algunas ideas adicionales:

- Utiliza otro recursos multimedia como videos y sonidos. Por ejemplo, en *Identifica a tus Compañeros* puedes alternar entre fotos y grabaciones de voz. Si usas más archivos de sonidos puedes crear una aplicación para identificar canciones o melodías!
- El codigo para comparar las respuestas es muy estricto, por ejemplo *"Raíz"* y *"raíz"* no se consideran iguales porque una letra es mayúscula y la otra no. Soluciona esto mediante una normalización del texto ingresado por el usuario, usando bloques de la sección *"Texto"*. Por ejemplo, puedes pasar el texto a minúsculas y remover todos los acentos.
- Crea una aplicación con preguntas de selección múltiple. Vas a tener que almacenar las posibles respuestas dentro de listas, por lo que tendrás que manejar una lista de listas. Utiliza el componente **SelectorDeLista** para que el usuario elija una respuesta.
- Combinando lo anterior, puedes crear una aplicación al estilo Quien Quiere ser Millonario, incorporar comodines, etc.

5. Material de Apoyo

Navegación de una Lista

¿Cómo se puede recorrer una lista con información? Ya sabemos que AppInventor provee el bloque **por cada ...** que permite procesar todos los elementos de una lista, uno por uno; pero que esencialmente los procesa todos a la vez. ¿Que pasa si tienes una aplicación como una trivia o un pase de diapositivas, en la cual el usuario controla el movimiento a través de la lista? Para programar este comportamiento, necesitas usar una variable como índice para registrar tu posición en la lista. Revisa los siguientes ejemplos.

Ejemplo: ¿Cómo creas un pase de diapositivas donde el usuario presiona un botón para ver la siguiente foto? En este ejemplo, el usuario presiona un botón para avanzar

en la secuencia de diapositivas que contienen fotos. Las imágenes corresponden a los archivos *foto1.jpg, foto2.jpg y foto3.jpg*, las cuales han sido subidas a la aplicación. La Figura 5.1 muestra el código de la aplicación.

Figura 5.1: Código aplicacion para pase de diapositivas.

La variable índice se de ne para mantener un registro de la posición de la figura actual que el usuario está viendo. Cuando el usuario presiona “Siguiente", se averigua si el índice ha ido muy lejos (más allá del largo de la lista). Si el índice es menor que el largo de la lista, que en este caso es 3, entonces el índice se incrementa y se pone la siguiente imagen de la lista en la propiedad **Imagen1.Foto**.

Veamos paso a paso este ejemplo. Asume que foto1.jpg se muestra cuando se inicia la aplicación. Cuando el usuario presiona “Siguiente", se revisa el índice y efectivamente es menor que el largo de la lista (sabemos que 1 < 3). Por lo tanto se incrementa el índice desde su valor inicial 1, al valor 2; y se selecciona y muestra la foto2.jpg.

La próxima vez que el usuario presiona “Siguiente", el índice es 2 por lo que sigue siendo menor que 3, as que se incrementa su valor a 3 y se muestra la tercera foto, foto3.jpg.

Cuando se presiona otra vez “Siguiente", el índice vale 3 por lo tanto el chequeo (3 < 3 no es verdad) falla. Por lo tanto los bloques dentro del condicional no se ejecutan,y nada pasa. De hecho, el usuario puede seguir presionando “Siguiente" mil veces, sin efecto alguno.

Ejemplo: ¿Cómo agregar un botón “Atrás" para mostrar las diapositivas anteriores? El código de este ejemplo, que se ve en la Figura 5.2 es similar al botón “Siguiente" anterior, pero ahora

el ndice se disminuye para volver a la diapositiva anterior. En este caso la condición chequea si el índice es mayor que 1; porque si ya es 1, ya se está mostrando la primera foto y no es posible ir más atrás.

Figura 5.2: Código para botón “Atrás".

Ejemplo: ¿Cómo comenzar nuevamente las diapositivas una vez llegado al final? En el ejemplo de la Figura 5.3, cuando el usuario presiona *“Siguiente"* cuando se está mostrando la última diapositiva, el *índice* se pone como 1 nuevamente, lo que hace que se “reinicie" el pase de diapositivas. El bloque poner **Imagen1.Foto** ahora esta afuera del bloque **si-sino** por lo tanto siempre se mostrar una foto diferente cuando el usuario presiona *“Siguiente"*.

Figura 5.3: Código para reiniciar una vez terminado el recorrido