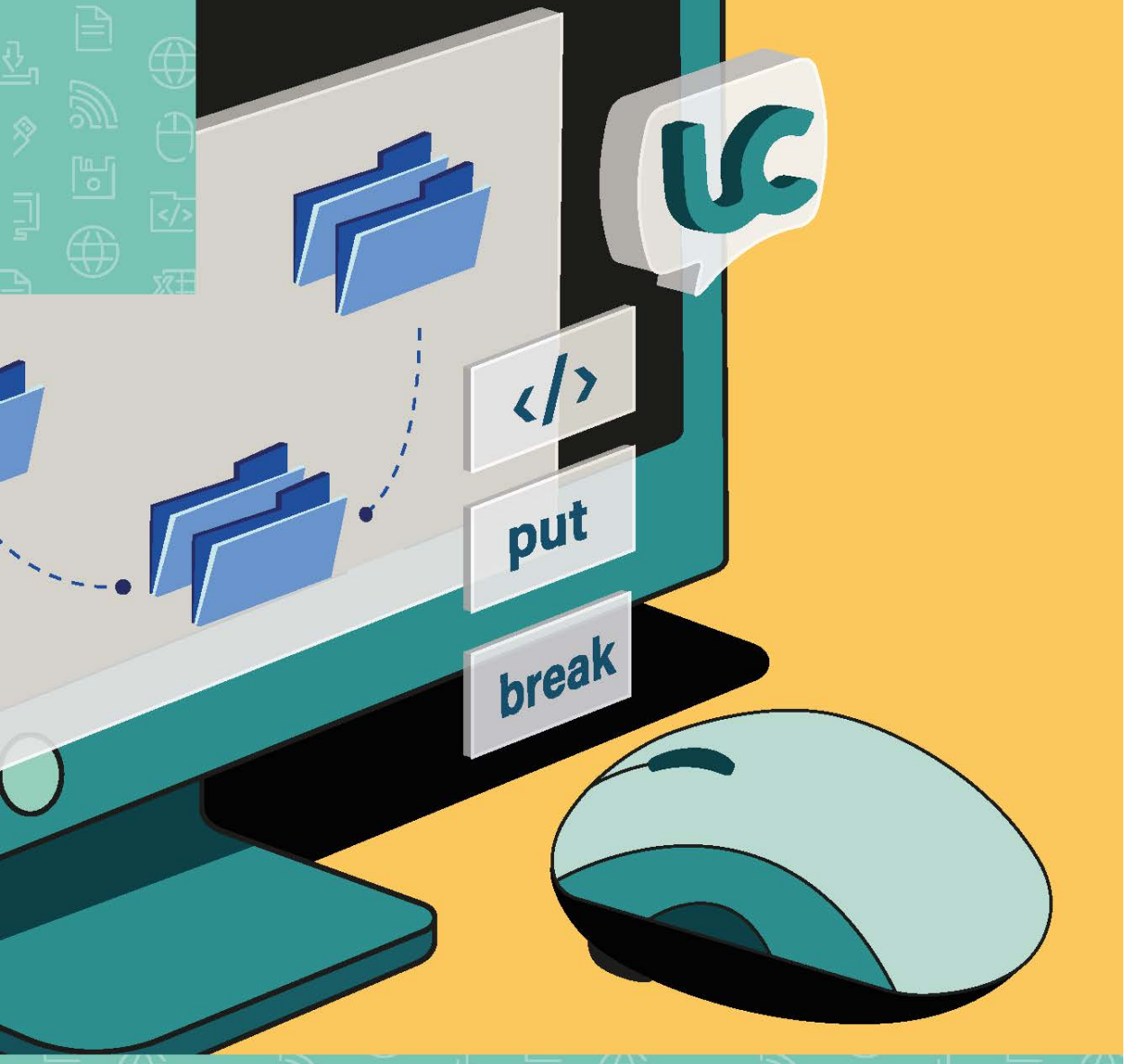


INFORMÁTICA

onceno grado

INFORMÁTICA

onceno grado

M. Sc. Ilien Pérez González
M. Sc. César Ramón Labañino Rizzo

EDITORIAL
PUEBLO Y EDUCACIÓN

Este material forma parte del conjunto de trabajos dirigidos al Tercer Perfeccionamiento Continuo del Sistema Nacional de la Educación General. En su elaboración participaron maestros, metodólogos y especialistas a partir de concepciones teóricas y metodológicas precedentes, adecuadas y enriquecidas en correspondencia con el fin y los objetivos propios de cada nivel educativo, de las exigencias de la sociedad cubana actual y sus perspectivas. Ha sido revisado por la subcomisión responsable de la asignatura perteneciente a la Comisión Nacional Permanente para la revisión de planes, programas y textos de estudio del Instituto Central de Ciencias Pedagógicas del Ministerio de Educación.

Edición y corrección:

- Lic. Yaimara Borges Forcades

Diseño de cubierta, ilustración y emplane:

- Instituto Superior de Diseño

Diseño:

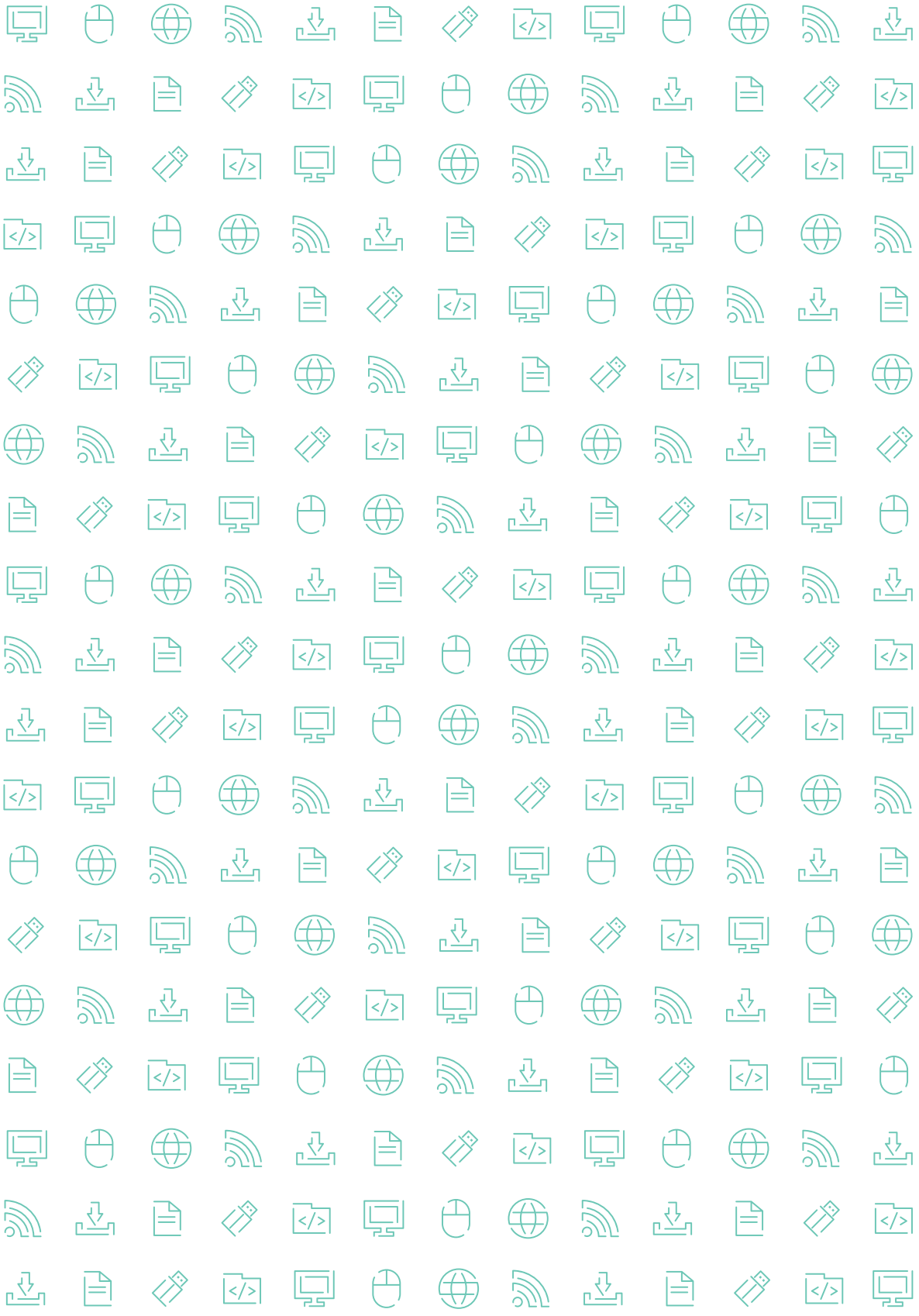
- Instituto Superior de Diseño (ISDi):
  Adriana Vigil Hernández ■ Alessandra Fuentes Tiel ■ Jennifer González Espinosa ■ Thalia Ibarra Villavicencio ■ Laura Ramos García ■ Ernesto Alejandro Gilart Ruiz ■ María Fernanda Lemus González ■ Aldahir Santana Guzmán ■ Litsary Zamora Rodríguez ■ Samira González González ■ Marian Ramos Rodríguez ■ Kamila Carpio Crespo ■ DCV María Paula Lista Jorge ■ M. Sc. Maité Fundora Iglesias ■ Dr. C. Ernesto Fernández Sánchez

ISBN 978-959-13-4817-3 (Versión impresa)

ISBN 978-959-13-4818-0 (Versión digital)

EDITORIAL PUEBLO Y EDUCACIÓN
Ave. 3.ª A No. 4601 entre 46 y 60,
Playa, La Habana, Cuba. CP 11300.
epueblo@epe.gemined.cu

ÍNDICE



Prólogo

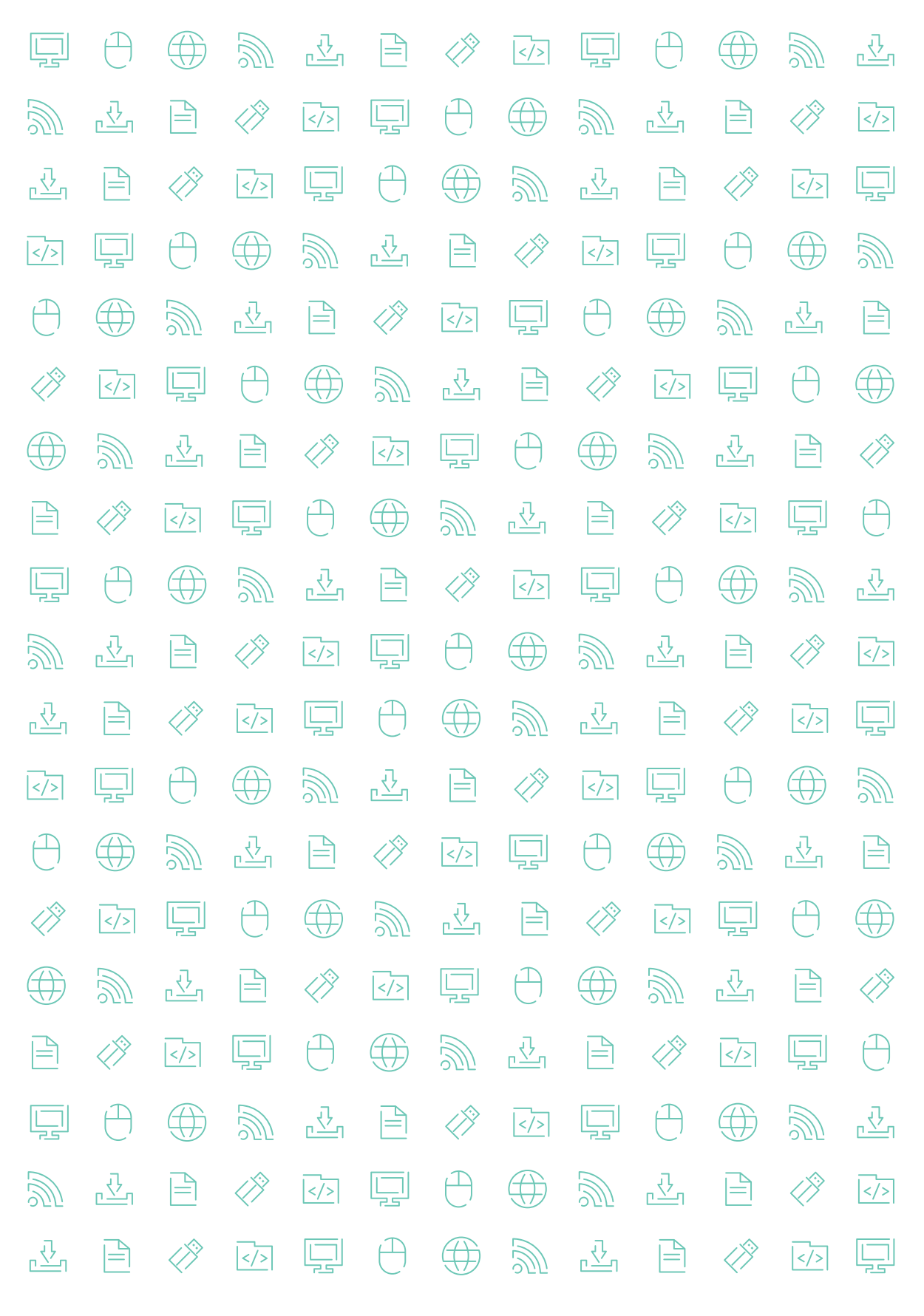
Estimado educando, ponemos en tus manos este libro de texto de Informática con el objetivo de que poseas un medio de enseñanza que aborde los contenidos del grado en una asignatura que, como sabes, es de suma importancia, tanto para el desarrollo de nuestro país como para el tuyo personal como futuro ciudadano.

El libro contiene dos capítulos: "Gestionando bases de datos" y "Profundizando en la programación". En el capítulo 1 se trata lo referente a los sistemas de gestión de bases de datos (SGBD) y su tema se expone desde la perspectiva del ***software*** libre mediante el SGBD SQLite. En este capítulo se abordan los conceptos básicos de bases de datos con el objetivo de preparar a los educandos para su posterior tratamiento desde el lenguaje de programación LiveCode.

En el capítulo 2 se propone un abordaje sistemático a la programación basada en objetos y dirigida por eventos. Se recuerdan los conceptos básicos relacionados con la lógica de programación, la representación de algoritmos y su implementación en LiveCode, donde se trabaja a partir de invariantes conceptuales de esta temática. Sin lugar a dudas, estos serán elementos que van a preparar al educando para implementar la solución de problemas de la vida práctica mediante cualquier lenguaje de programación. El capítulo concluye con el proceso de desarrollo desde la ingeniería de ***software*** y la realización de un proyecto integrador.

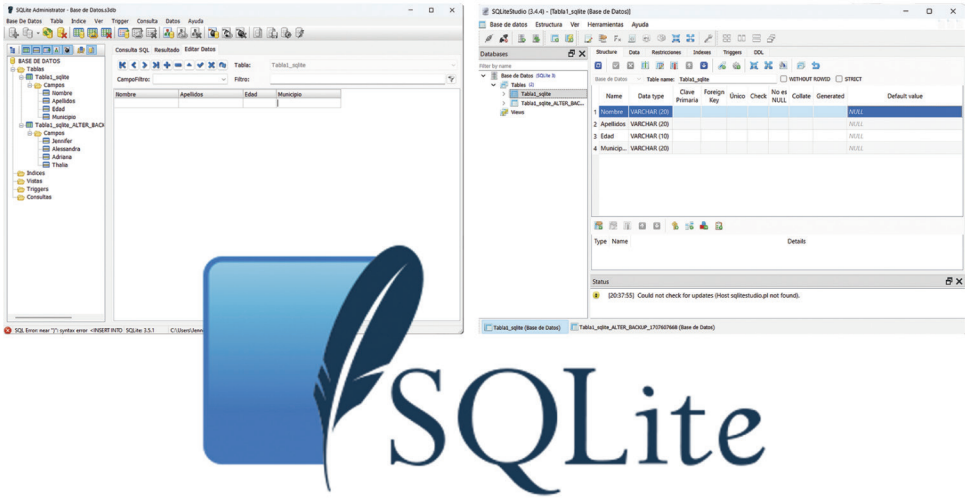
Esperamos que hagas un uso óptimo de este medio de enseñanza, el cual te puede servir de estímulo para que des pasos firmes en este apasionante mundo de la informática.

También, como autores, le dedicamos este libro a todas las muchachas que cursan el preuniversitario y, en especial, a nuestras hijas: Indira, Rocío, Rosalí, Nathalie y Thalía, para que al igual que Ada Byron rompan el estigma de que las mujeres no pueden ser programadoras.



CAPÍTULO 1

Gestionando bases de datos



Desde que surge la actividad humana, el hombre tuvo la necesidad de encontrar una forma para organizar y controlar sus recursos. Luego, ante el constante desarrollo de la información, que ha venido sucediendo durante siglos, se hizo necesario el perfeccionamiento de ese control de datos e informaciones, así como el agilizar su procesamiento.

El análisis de la capacidad que poseen los sistemas informáticos para almacenar, controlar y procesar datos permitió establecer el vínculo entre los datos, la información y el control, o sea, el principio de las *bases de datos.*

En la actualidad, las ***bases de datos*** son ampliamente usadas y aplicadas en cualquier campo del saber, ya que es el modo más popular de almacenamiento de la información. Debido al desarrollo tecnológico de disciplinas como la electrónica y la informática, la mayoría de estas bases de

datos están en formato digital (electrónico) ofreciendo un amplio rango de soluciones al problema de almacenar datos.

Hoy en día los temas relacionados con las bases de datos son parte de la formación profesional en diversas carreras de la Educación Superior y la Enseñanza posgraduada. Por la importancia que tiene el procesamiento automatizado de datos y su utilización en la programación, en esta primera unidad estudiarás la forma de crear y gestionar una ***base de datos*** utilizando el SQLite, como sistema completo de bases de datos que soporta múltiples tablas, índices, etcétera.

Entre los ejemplos de situaciones que generan la necesidad de controles y procesamiento de los datos, se pueden citar las siguientes: la reservación de un determinado paquete turístico; los datos que reflejan los daños al ecosistema por la actividad de la producción de bienes materiales; los datos del programa de salud para contabilizar la cantidad de personas afectadas por enfermedades crónicas como la hipertensión arterial, entre otras; los datos de los diferentes libros que posee la biblioteca de una escuela; entre otros tipos de datos que se pueden generar en diferentes esferas de trabajo.

¿Qué vas a aprender?

Conocerás sobre las bases de datos, sus tipos, conceptos fundamentales, objetos, forma de crearlas y gestionarlas.

¿Para qué me sirve?

Para almacenar información, reduciendo la redundancia en los datos almacenados al permitir la compartición de dicha información manteniendo la integridad de los datos.

¿Qué debo saber?

Debes conocer: ¿qué es un dato?, los elementos del diseño de una base de datos, ¿qué sistemas de aplicación permiten gestionar las bases de datos?

1.1 Introducción a los sistemas de gestión de base de datos (SGBD)

El archivo que se crea por sí mismo no constituye una base de datos, es la forma en la que está organizada la información la que da origen a la ***base de datos.***

Definición

¿Qué es una ***base de datos***? Es un conjunto de datos organizados e interrelacionados entre sí, que tienen una estructura lógica; los cuales son almacenados con un carácter más o menos permanente en un sistema informático. También se puede acceder a ellos con facilidad con diferentes propósitos.

Algo de historia

El término base de datos fue acuñado por primera vez en 1963, en un simposio celebrado en California.

Definición

¿Qué *es* ***un Sistema de Gestión de Bases de Datos****?* Un *Sistema de Gestión de Bases de Datos* (SGBD) es un *software* que permite almacenar, organizar y manipular gran cantidad de datos en una o varias bases de datos integradas, manejando todas las solicitudes de acceso formuladas por los usuarios desde diferentes puntos de vista

Dicho de otra forma, un SGBD es el ***software*** dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan, permitiendo describir los elementos de datos con su estructura, sus interrelaciones y sus validaciones.

Estos SGBD, creados para operar con las bases de datos, reúnen las siguientes características generales:

- Independencia entre los datos y los programas de aplicación.
- Mínima redundancia de los datos.
- Rapidez de acceso a los datos.
- Independencia entre los datos y los dispositivos físicos.
- Máxima garantía de integridad de los datos.
- Máxima protección contra fallos y errores.
- Control centralizado.

Hoy en día podemos encontrar una gran variedad de SGBD como representantes del ***software*** libre, por ejemplo: PostgreSQL, Apache Derby, SQLite, MariaDB entre otros, y como representantes del ***software*** propietario: Oracle, Open Access, Microsoft Access, Fox Pro, etcétera.

En un sistema informático existen diferentes formas de almacenar la información, esto da lugar a distintos modelos de organización de la base de datos, entre ellos están: jerárquico, red, multidimensional, transaccional, orientado a objeto y relacional.

En este grado estudiaremos el ***sotftware*** libre SQLite, que es un SGBD relacional. Las bases de datos relacionales tienen como idea fundamental el uso de relaciones, donde se ofrecen muchos tipos de procesos de datos, como: simplicidad y generalidad, facilidad de uso para el usuario final, períodos cortos de aprendizaje, etcétera, y la información puede ser recuperada o almacenada mediante consultas que ofrecen una amplia flexibilidad para administrar la información.

Después de conocer que los conceptos de base de datos y SGBD nos ayudan a gestionar nuestros datos, al proporcionarnos una estructura eficiente para almacenar y recuperar la información, se puede afirmar que el paso clave para un almacenamiento y recuperación eficiente de los datos es el proceso de planificación, es decir, antes de interactuar con el SGBD se debe realizar un trabajo de mesa con lápiz y papel.

El primer paso en el proceso del proyecto es reconocer qué estructura externa tienen los datos, la tarea que el SGBD debe realizar con los datos y la información que necesitamos gestionar.

Por ejemplo, una silla es una entidad u objeto, así como un automóvil, un empleado, un profesor, un educando, que son cosas concretas; pero también los datos pueden ser algo no tangible, como un suceso cualquiera, una cuenta de ahorro, o un concepto abstracto.

Definición

Campo o atributo: es la unidad menor de información sobre un objeto (almacenada en la base) y representa una de las propiedades de dicho objeto (por ejemplo, el color). Sin embargo, es importante saber distinguir entre el *nombre o tipo de atributo* y el *valor del atributo*, ya que un nombre de atributo puede tomar diferentes valores sobre un cierto conjunto que se denomina *dominio.*

A un valor de un atributo determinado o definido en el dominio dado, en un cierto momento del tiempo, se le denomina ***ocurrencia del atributo.*** Por ejemplo:

Atributo ***Color***

Dominio ***{Azul, Rojo, Verde,...}***
Ocurrencia ***Rojo***

Ahora bien, debes tener presente que después de definido el campo es momento del segundo paso, que será completar los registros de acuerdo con la información que debemos procesar, a continuación se verá que son los registros o artículos.

Definición

Una colección de campos asociados es un ***artículo*** o ***registro,*** y representa un objeto con sus propiedades.

Por ejemplo: el nombre o tipo de artículo puede ser "Profesor", y este puede estar formado por los siguientes tipos de campos o atributos:

NUM_IDENT: número de identidad del profesor
NOM_PROF: nombre del profesor
CAT_DOC: categoría docente del profesor
DPTO: departamento docente al que pertenece el profesor

Una ***ocurrencia*** de este artículo puede ser:
45112801731 Hdez Roberto PA Computación.

El tercer paso es el de los datos propiamente dichos, representados mediante cadenas de caracteres o de bits. En este paso es necesario tener en cuenta la diferencia entre tipo de dato y valor del dato.

Ahora vamos a pasar a estudiar algunos de los elementos de una base de datos relacional. El primer elemento es la tabla, donde se va a almacenar el conjunto de datos. Este será el objeto más importante de nuestra base de datos y estará formado por columnas que corresponden a cada uno de los campos o atributos, las cuales deben tener un nombre. Por otro lado, las filas conforman lo que llamamos ***registros*** o ***artículos.***

Pero una base de datos no está compuesta solo por ***las tablas;*** otros elementos ayudan a hacer más viable el procesamiento de los datos por los SGBD, ellos son: las ***consultas,*** los ***formularios*** y los ***informes.***

Definición

Las ***consultas*** son herramientas que permiten hacer flexible el tratamiento del conjunto de datos, respondiendo a una pregunta que se le hace a la base de datos; pueden utilizarse para ver y analizar los datos de las tablas de diferentes maneras.

Definición

Los ***formularios*** son las herramientas para hacer más asequibles los recursos de los SGBD, facilitan la entrada y salida de información digital por pantalla y la opción de impresión en papel, al tener la posibilidad de establecer un formato en pantalla para la captura y lectura de datos, y su posible impresión en papel, acorde a sus necesidades.

Definición

Los ***informes*** son herramientas que permiten preparar los registros de la base de datos de forma personalizada para imprimirlos.

Ya se han analizado los conceptos fundamentales que debes conocer sobre las bases de datos y los sistemas de gestión de bases de datos. Ahora pasaremos a la creación de bases de datos sencillas utilizando como herramienta el SQLiteAdministrator, un tipo de SGBD. Aplicación sencilla que permite administrar bases de datos SQLite.

1.2 Creando bases de datos con SQLiteAdministrator

En este epígrafe se comenzará analizando las características de SQLite. SQLite es un SGBD muy similar al conocido Access del mundo Office. Es una biblioteca en C que implementa un motor de bases de datos SQL. Fue escrito por el Dr. Richard Hipp en el año 2000, como un proyecto de código abierto, pero a diferencia de este posee una serie de ventajas que explicaremos a continuación. Para comenzar es multiplataforma y cumple con los estándares (en su mayoría) SQL92, por lo que es ideal para trabajar con volúmenes medianos o pequeños de información de manera

ágil y eficiente, aunque sus diseñadores plantean que posibilita el manejo de bases de datos de 2 terabytes sin mayores inconvenientes. Además, se debe tener claro que SQLite es una base de datos embebida, por lo que no requiere de un programa adicional para ejecutarse, solo basta con agregar el archivo de la base de datos al instalador de nuestra aplicación para que sea funcional en cualquier máquina donde se instale.

Este es un sistema completo de bases de datos que soporta múltiples tablas, índices, triggers y vistas. El formato de su base de datos es multiplataforma e, indistintamente, puede utilizar el mismo archivo en sistemas de 32 bits y 64 bits, un archivo que se puede almacenar en un único fichero. También emplea registros de tamaño variable, de forma tal que utiliza en el disco el espacio que es realmente necesario en cada momento.

Esta es una herramienta que tiene un código fuente entendible y accesible para programadores promedio, aunque para explotar el sistema no hay que ser programador. Todas sus funciones y estructuras están bien documentadas y no necesita un proceso desde un servidor, ya que lee y escribe directamente sobre archivos que se encuentran en el disco duro.

Es importante destacar que existe un programa independiente de nombre SQLite, que puede ser utilizado para consultar y gestionar los ficheros de base de datos SQLite. También sirve como ejemplo para la escritura de aplicaciones utilizando la biblioteca SQLite.

SQLite se puede usar en modo ventana de comandos (***Shell***) o embebido en aplicaciones de código (es decir, casi con cualquier lenguaje de programación directamente o indirectamente), por ejemplo, C, C++, Bash, etcétera. Además, se puede utilizar con OpenOffice mediante la aplicación de los manipuladores ODBC adecuados.

En la actualidad gestionar las bases de datos SQLite se hace muy fácil con SQLiteAdministrator, herramienta totalmente gratuita y funcional en todos los operativos hasta la fecha; además, tiene el agregado de que se puede configurar el idioma español. Esta aplicación será la que emplearás para la gestión de tus bases de datos, pero también pudiera hablarse de otras herramientas para la gestión de base de datos SQLite, como: SQLiteManagent, SQLiteExpert, entre otras. También es importante que conozcas que navegadores como Firefox poseen plugins para gestionar bases de datos SQLite. En todos ellos los principios de funcionamiento son los mismos.

Ahora comenzaremos analizando la interfaz de trabajo y las principales opciones del SQLiteAdministrator.

Lo primero que debes hacer es ejecutar la aplicación, que como no necesita ser instalada en nuestro ordenador, te resultará muy fácil, pues solo deberás abrir la carpeta que contiene el archivo ejecutable llamado ***SQLiteadmin.exe*** y dar doble clic sobre él.

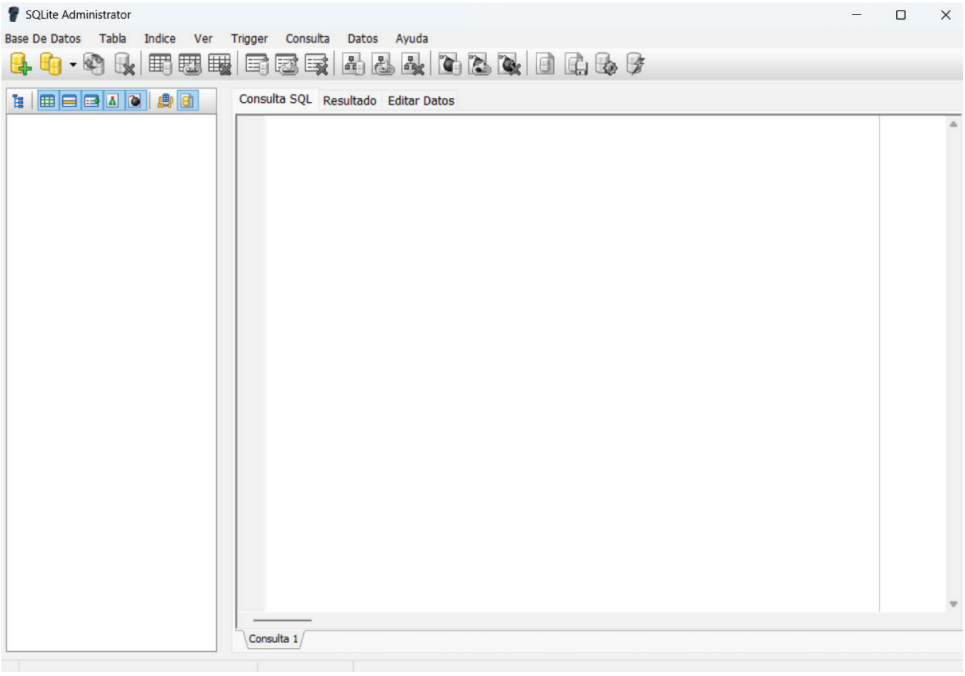


Fig. 1.1

Al ejecutar la aplicación, lo primero que encontrarás será la pantalla principal (figura 1.1). En esta pantalla principal puedes observar, como en todas las aplicaciones, la barra de herramientas (figura 1.2) y, a continuación, dos áreas básicas: a la izquierda, un navegador de objetos y a la derecha, se muestran las pestañas: "consultas SQL", "resultados" y "Editar datos" (figura 1.3).

Fig. 1.2

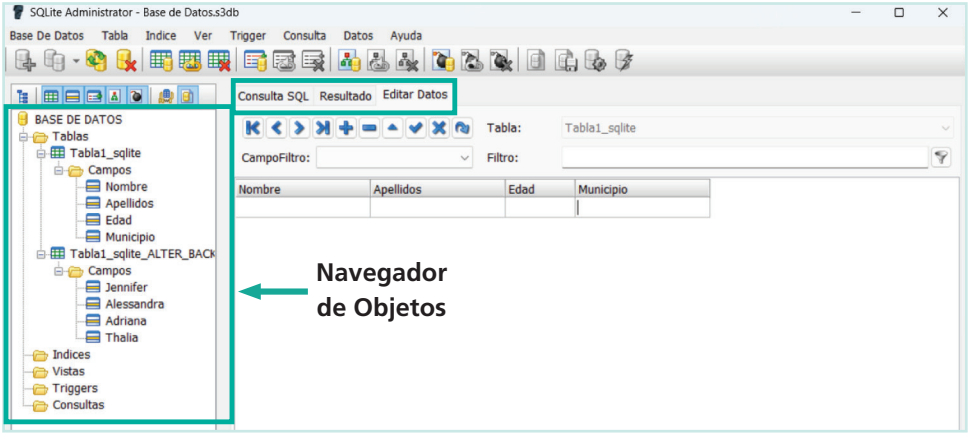


Fig. 1.3

En la parte superior también encontraremos ocho menús que te permitirán realizar todas las operaciones que necesites. El primero es el menú ***Base de datos*** (figura 1.4), donde encontrarás opciones similares a las de otros sistemas de aplicación como: crear una nueva base de datos, abrir una existente, acceder a las utilizadas recientemente, limpiar, cerrar la base de datos, migrar a SQLite3, SQL Base de datos y salir de la aplicación.

En los menús ***Tabla, Índice, Ver*** y ***Trigger*** (figura 1.5) aparecen funciones como: nuevo, editar y eliminar opciones básicas para realizar con los objetos de la base de datos.

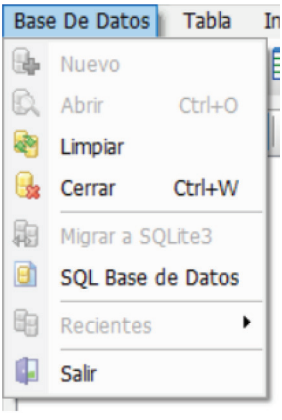


Fig. 1.4

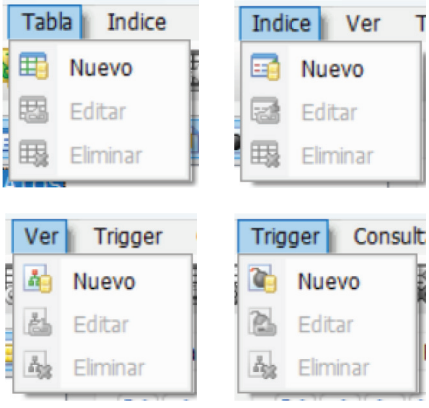


Fig. 1.5

En el menú ***Consultas (***figura 1.6) podrás: agregar, abrir, guardar, eliminar, ejecutar con o sin resultado, utilizar en "vista nueva" o utilizar en

"Trigger nuevo" las consultas a tu base de datos mediante líneas de código. En el menú ***Datos*** (figura 1.7) podrás exportar e importar datos.

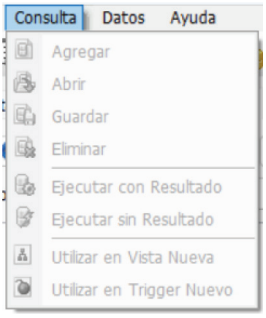


Fig. 1.6

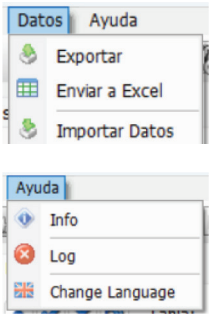


Fig. 1.7 y 1.8

Por último, como es costumbre en todas las aplicaciones aparece el menú ***Ayuda*** (figura 1.8), donde podrás encontrar una información general de la aplicación, un registro de errores y tendrás la posibilidad de cambiar el idioma a la aplicación.

Después de haber analizado los elementos fundamentales de la interfaz de trabajo, se te propone pasar a la creación de una base de datos y algunos de sus objetos, a partir de la solución práctica del siguiente ejercicio.

Práctica guiada

En tu centro escolar la secretaria docente necesita controlar los datos personales de todos los educandos de la escuela, dígase: nombre, apellido, dirección, edad, fecha de nacimiento, Carnet de Identidad (CI) y teléfono, también el grado que cursa y el grupo en el que se encuentra matriculado el educando. La tabla que aparece a continuación muestra los datos de 10 educandos que son matricula de la escuela (tabla 1.1). Crea una base de datos que permita la gestión y el control de esta información por la secretaria docente. Además, crea un informe donde aparezcan solo los educandos que tengan 15 años cumplidos.

Recuerda que...

Una buena práctica que se debe seguir en la creación de una base de datos, tiene en cuenta los siguientes aspectos:

- Es recomendable que los nombres de los ficheros de las ***bases de datos*** no tengan espacios, tildes ni ***ñ***, eso evitará conflictos en el acceso a la base de datos, y, sobre todo, en Linux.
- Lo aconsejable es que la base de datos no se guarde en la misma partición del sistema operativo para evitar la pérdida de la información, si este deja de funcionar correctamente y es necesario reinstalar.

Tabla 1.1

Nombre	Apellido	Dirección	Edad	Fecha de nacimiento	CI	Teléfono	Grado	Grupo
Anthony Harold	Armenteros	204 A # 204A03 apto.10 / 25A y 27. La Coronela.	16	040220	04022066622	76431837	12mo.	10
Alejandro Antonio	Borrero	226 # 27A26/ 27 y 31. La Coronela.	15	020708	02070868044	72764258	10mo.	5
Tomás Enrique	Cala	5ta # Edif. 9 apto. 3. Aldabó.	15	020129	02012968068	52705743	10mo.	3
Helen	Calvo	Ave. 67 A # 15816 int. / 158 y 160.	15	020913	02091367451	72669432	10mo.	5
Glenda Yisabel	Calvo	Bernaza 164 Hab. 12-13 / Lamparilla y Tte. Rey. Habana Vieja	16	030917	03091767554	77925654	11no.	8
Patricia	Corzo	San Francisco 13321 / Massip y San Ignacio. Bellavista.	14	020208	02020866571	72248590	10mo.	1
Odette	Esmoris	Santa Beatriz 10 apto. 2 / Villoldo y Porvenir. Vibora Park.	16	031111	03111166818	758624595	11no.	8
Laurent Lil	Fernández	Ave. Cacahual. El Palmar.	15	020727	02072775213	72096358	10mo.	3
Marianne	Fonseca	Corrales 10 Apto. 23 / Zulueta y Egido.	15	020715	02071566572	72085634	10mo.	1
Janiel	Gainza	61 # 11813 int. 1 / 118 y 120.	17	040309	04030966967	78258606	12mo.	10

Para solucionar este ejercicio, se debe comenzar por la parte más importante que es definir ***en papel*** el diseño de nuestra base de datos. Esto consiste en analizar el problema y establecer cuáles serán los campos que se deben crear, qué nombres recibirán y los tipos de datos que se almacenarán en cada uno de ellos, así como sus propiedades.

En nuestro caso, de la lectura y análisis de la situación propuesta se puede concluir que la base de datos podría nombrarse "Control de educandos", esta tendría una tabla nombrada "Datos Personales", y dicha tabla estaría formada por los siguientes campos:

- Nombre:
- Apellido:
- Dirección particular:
- Edad:
- Fecha de nacimiento:
- C. I.:
- Teléfono:
- Grado:
- Grupo:

Ahora bien, ¿qué tipo de datos nos brinda SQLite para almacenar nuestros datos en cada uno de estos campos? Lo primero que debes saber es que el tipo de datos SQLite es un atributo que especifica el tipo de datos de cualquier objeto. Cada columna, variable y expresión, tiene relación con el tipo de datos en SQLite y el tipo de datos de un valor se asocia con el valor en sí, no con su contenedor.

Cada valor almacenado en una base de datos SQLite tiene una de las siguientes clases de almacenamiento (tabla 1.2).

Tabla 1.2

Clase de almacenamiento	Descripción
NULL	El valor es un valor nulo.
INTEGER	El valor es un entero con signo, almacenado en 1, 2, 3, 4, 6, u 8 bytes, dependiendo de la magnitud del valor.
REAL	El valor es un valor de punto flotante, almacenado como un número de coma flotante de 8 bytes (IEEE).

TEXT	El valor es una cadena de texto, almacenada utilizando la codificación de la base de datos (UTF-8, UTF-16BE o UTF-16LE).
BLOB	El valor es una blob (gota) o masa de datos, almacenada exactamente como se ingresó.

La clase de almacenamiento de SQLite es ligeramente más general que un tipo de datos. La clase de almacenamiento INTEGER, por ejemplo, incluye seis diferentes tipos de datos enteros de diferentes longitudes.

SQLite apoya el concepto de ***typeaffinity*** en columnas. Cualquier columna todavía puede almacenar cualquier tipo de datos, pero la clase de almacenamiento preferido para una columna se llama ***affinity.*** A cada columna de una tabla en una base de datos SQLite3, se le asigna una de las siguientes afinidades (tabla 1.3).

Tabla 1.3

Afinidad	Descripción
TEXT	Almacena todos los datos de la columna utilizando las clases de almacenamiento NULL, texto, o BLOB.
NUMERIC	Esta columna puede contener valores utilizando las cinco clases de almacenamiento.
INTEGER	Se comporta igual que una columna con afinidad NUMERIC, con una excepción en una expresión CAST.
REAL	Se comporta como una columna con afinidad NUMERIC, excepto que obliga a valores enteros en representación del punto flotante.
NONE	Una columna con afinidad NONE, que no prefiere una clase de almacenamiento sobre otro y no hace ningún intento de coaccionar a los datos de una clase de almacenamiento a otro.

La siguiente tabla te relaciona algunos tipos de datos con una determinada afinidad (tabla 1.4).

Tabla 1.4

Tipo de datos	Afinidad
■ INT ■ INTEGER ■ TINYINT ■ SMALLINT ■ MEDIUMINT ■ BIGINT ■ UNSIGNED BIG INT ■ INT2 ■ INT8	INTEGER
■ CHARACTER(20) ■ VARCHAR(255) ■ VARYING CHARACTER(255) ■ NCHAR(55) ■ NATIVE CHARACTER(70) ■ NVARCHAR(100) ■ TEXT ■ CLOB	TEXT
■ BLOB ■ no datatypespecified	NONE
■ REAL ■ DOUBLE ■ DOUBLE PRECISION ■ FLOAT	REAL
■ NUMERIC ■ DECIMAL(10,5) ■ BOOLEAN ■ DATE ■ DATETIME	NUMERIC

También debes saber que SQLite no tiene una clase de almacenamiento de Boolean separada, en lugar de ello, los valores booleanos se almacenan como números enteros 0 (false) y 1 (true); tampoco tiene una para almacenar fechas y/o tiempos, pero es capaz de almacenar fechas y horas como valores de texto, reales o enteros (tabla 1.5).

Tabla 1.5

Clase de almacenamiento	Formato de fecha
TEXT	Una fecha en un formato como "AAA-MM-DD HH:MM:SS.SSS".
REAL	El número de días desde el mediodía en Greenwich el 24 de noviembre de 4714 AC.
INTEGER	El número de segundos desde el 1970-01-01 00:00:00 UTC.

Bueno, después de esta explicación, se puede decir que la tabla de nuestra base de datos pudiese quedar de la siguiente forma:

- Nombre. Sería un campo de tipo Texto (Text) y no puede estar vacío.
- Apellidos. Sería un campo de tipo Texto (Text) y no puede estar vacío.
- Dirección Particular. Sería un campo de tipo Texto (Text).
- Edad. Sería un campo de tipo Entero (Integer).
- Fecha de Nacimiento. Sería un campo de tipo Fecha (Date).
- C. I. . Sería un campo de tipo Texto (Text) y no puede estar vacío.
- Teléfono. Sería un campo de tipo Entero (Integer).
- Grado. Sería un campo de tipo Texto (Text).
- Grupo. Sería un campo de tipo Texto (Text).

Ya se ha analizado el ejercicio y establecido el diseño de nuestra base de datos, ahora pasaremos a su creación en SQLiteAdministrator.

Como es lógico, comenzarás por abrir la aplicación y en el menú ***Base de datos*** vas a seleccionar la opción "Nuevo" o simplemente darás clic en el ícono de acceso rápido. Seguidamente te encontrarás en la ventana

donde deberás indicar el "nombre" y la "ubicación de la base de datos". En este caso, la nombraremos "Control de Educandos" y la ubicaremos en "Mis documentos". Se terminará el procedimiento haciendo clic en "Guardar" (figura 1.9).

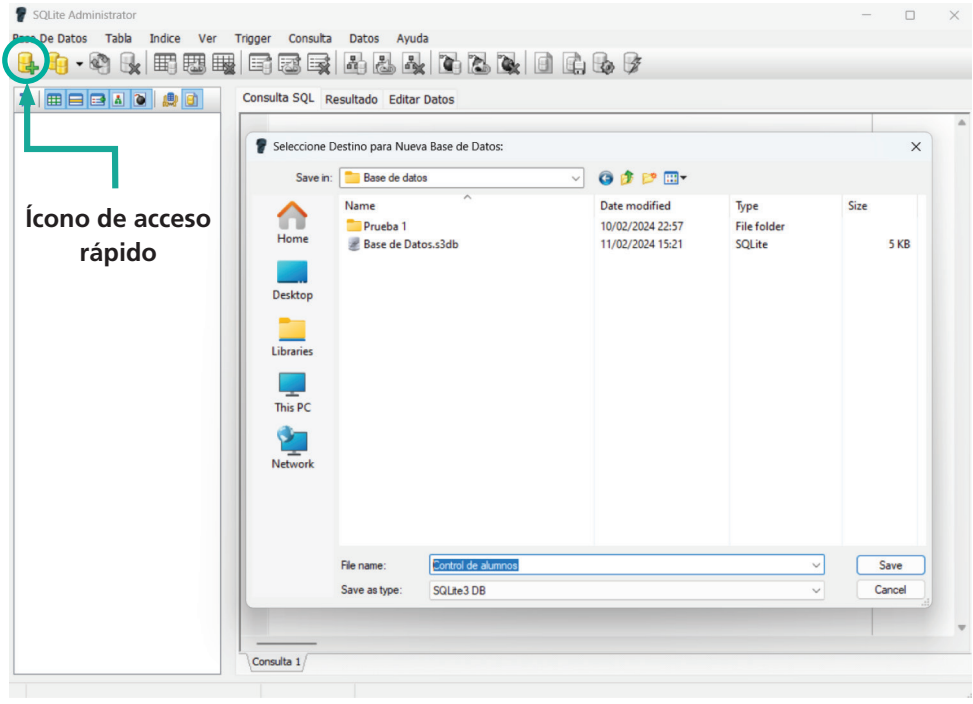


Fig. 1.9

Es importante que sepas que en los nombres de tablas y sus campos no puedes utilizar: acentos, números, espacios, caracteres especiales, ni el sufijo "sqlite". Además, SQLiteAdministrator nos ofrece solo dos tipos de extensiones para crear una base de datos SQLite, las cuales son:

- SQLite3 db: corresponde a la versión 3 de SQLite, la extensión del archivo generado es (.s3db)
- SQLite2 db: corresponde a la versión 2, que poco se usa actualmente, la extensión del archivo generado es (.sdb)

Seguidamente, aparecerá frente a ti la pantalla donde se muestran los diferentes elementos de la base de datos y en el extremo izquierdo de la ventana, te aparecerá una notificación diciendo que la base de datos ya fue creada (figura 1.10).

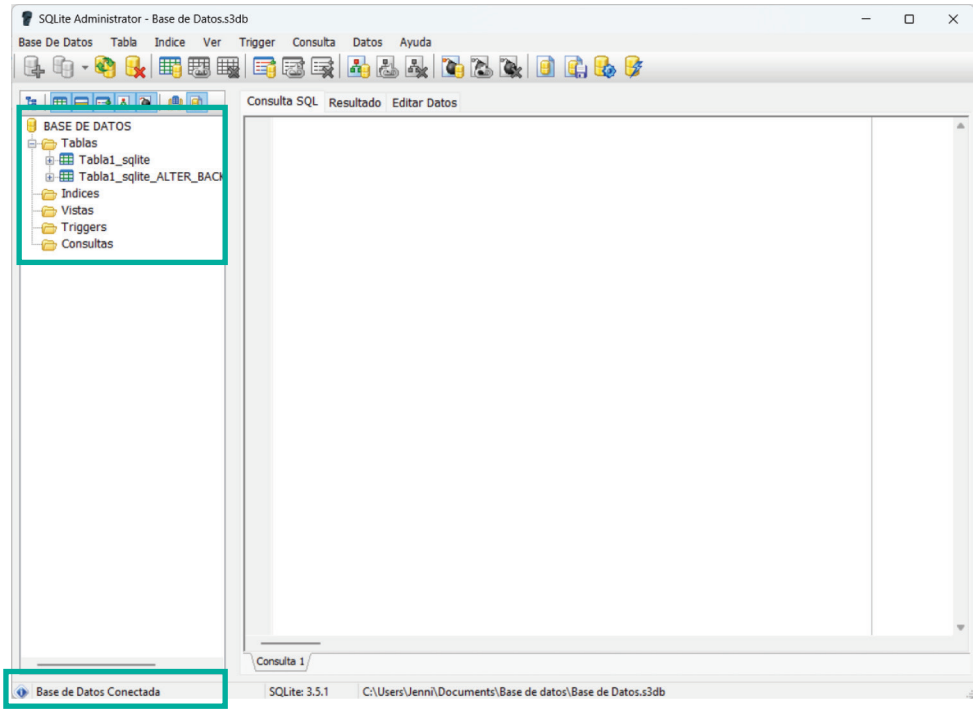


Fig. 1.10

Ya una vez creada nuestra base de datos, ha llegado el momento de crear nuestra tabla, esto lo haremos seleccionando en el menú ***Tabla*** la opción "Nuevo". Inmediatamente aparecerá una nueva ventana donde asignaremos el nombre a la tabla, que como recordarás se propuso que fuera "Datos personales" (figura 1.11). Como habrás observado, en esta ventana aparecen seis botones que nos permitirán: agregar, editar o eliminar campos, establecer la llave principal, crear y cancelar la operación.

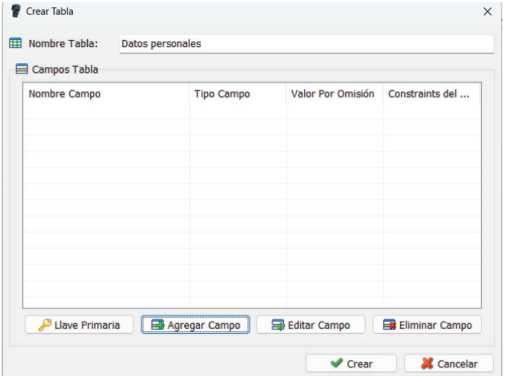
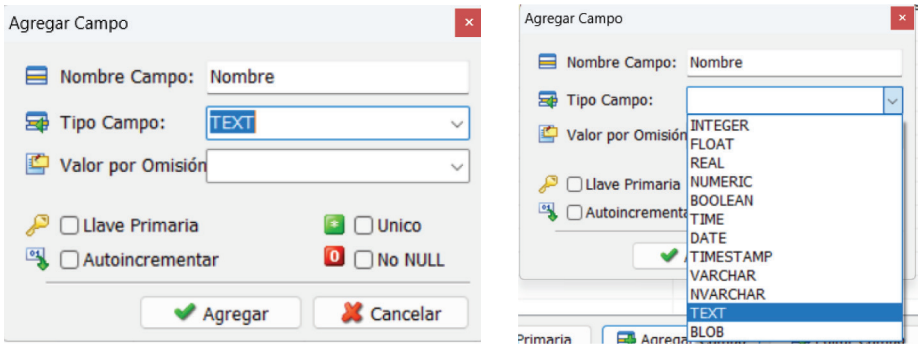


Fig. 1.11

Con la tabla creada, pasaremos ahora a la adición de cada uno de los campos. Esto se hará dando clic en el botón "Agregar campo". Al ejecutar

esta acción aparecerá una nueva ventana donde se va a establecer el nombre del campo, su tipo y propiedades (figura 1. 12 a, b). Se comenzará por el campo "Nombre" donde definiremos como tipo de dato "texto" y la condición que "no sea vacío".



a b

Fig. 1.12

Repetirás el procedimiento tantas veces como campos necesites crear en tu tabla. Al concluir debes dar clic en el botón "Crear" para hacer efectiva la creación de esta. Al hacerlo, aparecerá inmediatamente la tabla creada en el panel de navegación (figura 1.13). No olvides que puedes crear todas las tablas que necesites.

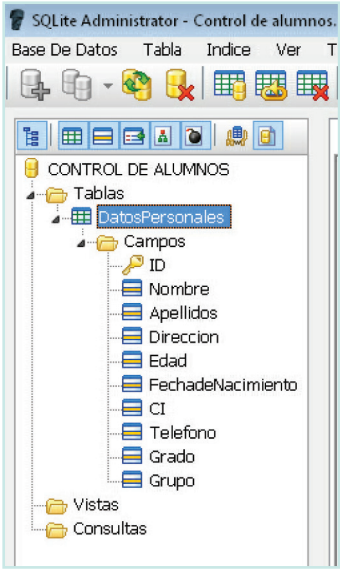


Fig. 1.13

A continuación, vas a agregar los registros a nuestra tabla. Para hacerlo deberás seleccionar la tabla y luego dar clic en la pestaña "Editar datos". Inmediatamente te aparecerá la estructura completa de la tabla y comenzarás a adicionar cada registro dando doble clic en el campo donde agregarás la información (figura 1.14).

Para que este trabajo de manipular los registros sea más sencillo, tenemos una barra de herramientas con varios botones que nos permiten: agregar, modificar y eliminar registros de manera más sencilla (figura 1.15).

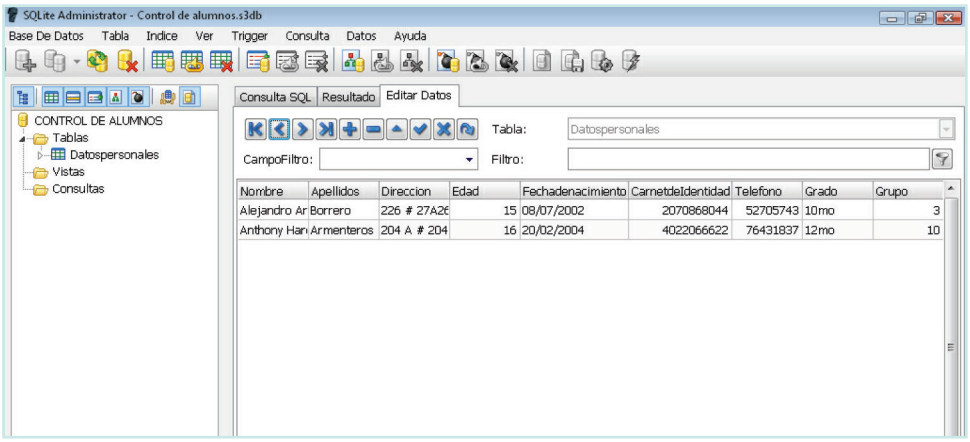


Fig. 1.14

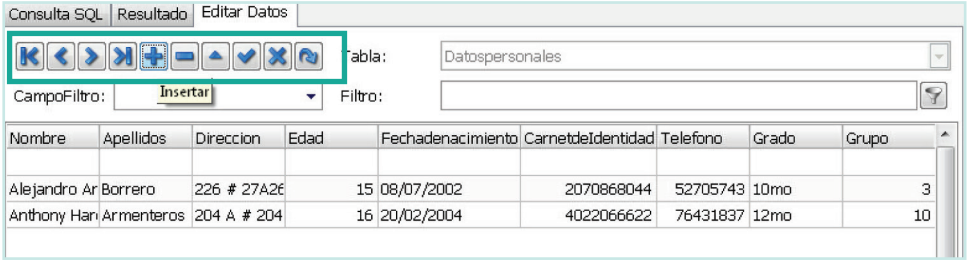


Fig. 1.15

Bueno, hasta aquí los elementos básicos para la creación de base de datos con SQLiteAdministrator, ahora se te propone que realices las siguientes actividades para que desarrolles habilidades en la creación de bases de datos.

Actividad práctica 1

1. En tu escuela se desea controlar la bibliografía existente como fondo bibliotecario teniendo en cuenta: el título de la publicación, autor, tipo de publicación, otros autores, cuidad de edición, editorial, año, cantidad de páginas, clasificación y precio. Crea una base de datos que permita solucionar este problema y muestra cuántas publicaciones pertenecen a la Editorial Pueblo y Educación.

Nota: para solucionar el ejercicio ve a la biblioteca de tu escuela y recopila toda la información necesaria para introducir al menos diez registros en tu tabla.

Actividad práctica 2

1. Una empresa de taxis de Ciudad de La Habana ofrece servicio a varios hospitales de esa ciudad. En la empresa trabajan choferes, de los cuales se conoce su número de identidad, nombres y años de servicio. De los taxis de la empresa se tienen los siguientes registros: la chapa, el número del motor, la marca y el modelo. Un taxi puede ser conducido por diferentes choferes en distintos momentos, pero un chofer siempre conduce el mismo taxi. Además, se conoce la cantidad de kilómetros totales recorridos por un chofer en su taxi. De cada hospital se conoce su nombre, tipo y dirección. Un chofer le brinda servicios (realiza viajes) a distintos hospitales y a un hospital le brindan servicios distintos choferes, así como se tiene información de la cantidad de viajes realizados por cada chofer a cada hospital. Crea la base de datos que permita controlar la información relacionada con los taxis, sus choferes y rutas.

Comprueba lo aprendido

1. En la siguiente sopa de letras encuentra los conceptos correspondientes a las siguientes definiciones:

- Conjunto de datos organizados e interrelacionados entre sí, que tienen una estructura lógica; los cuales son almacenados con carácter más o menos permanente en un sistema informático.
- Es la unidad menor de información sobre un objeto (almacenada en la base) y representa una de las propiedades de dicho objeto (por ejemplo, el color).
- Cosas o elementos que existen y están bien diferenciados entre sí, que poseen propiedades, y entre los cuales se establecen relaciones.
- Una colección identificable de campos asociados que representan un objeto con sus propiedades.

O	K	Á	M	V	K	B	E	Ú	E	L
C	L	D	R	R	V	A	B	Q	U	Y
A	D	P	D	Ñ	M	S	Z	Ü	É	Y
M	X	Ó	O	B	J	E	T	O	O	U
P	Ü	Í	B	C	W	D	B	O	W	Ñ
O	P	E	É	Í	R	E	I	X	Ú	L
W	O	G	Z	H	B	D	Ñ	L	É	Y
J	B	Y	Y	W	S	A	Z	R	H	F
Z	R	E	G	I	S	T	R	O	F	G
Ü	J	F	K	A	P	O	P	V	N	Ñ
Z	N	K	Q	T	F	S	Y	É	F	Ü

2. Descubre la palabra oculta si sabes que su significado es el siguiente: "***Software*** que permite almacenar, organizar y manipular gran cantidad de datos en una o varias bases de datos integradas, manejando todas las solicitudes de acceso, formuladas por los usuarios desde diferentes puntos de vista a la vez".

3. En una tienda se desean controlar las ventas y se conoce:

- Código de cada mercancía
- Descripción de cada mercancía
- País de procedencia de cada mercancía
- Área de moneda de cada país
- Precio de venta de cada mercancía según su calidad

a) Crea la base de datos que pueda controlar las ventas de la tienda.

4. En un círculo infantil se desea controlar la actividad de vacunación de los niños. Para ello se tiene la siguiente información.

Paracadaniño	Paracadavacuna
Númerodeidentidad	Nombrequelaidentifica
Nombre	Países deprocedencia
Fechadenacimiento	Condicionesdealmacenamiento
Direcciónparticular	

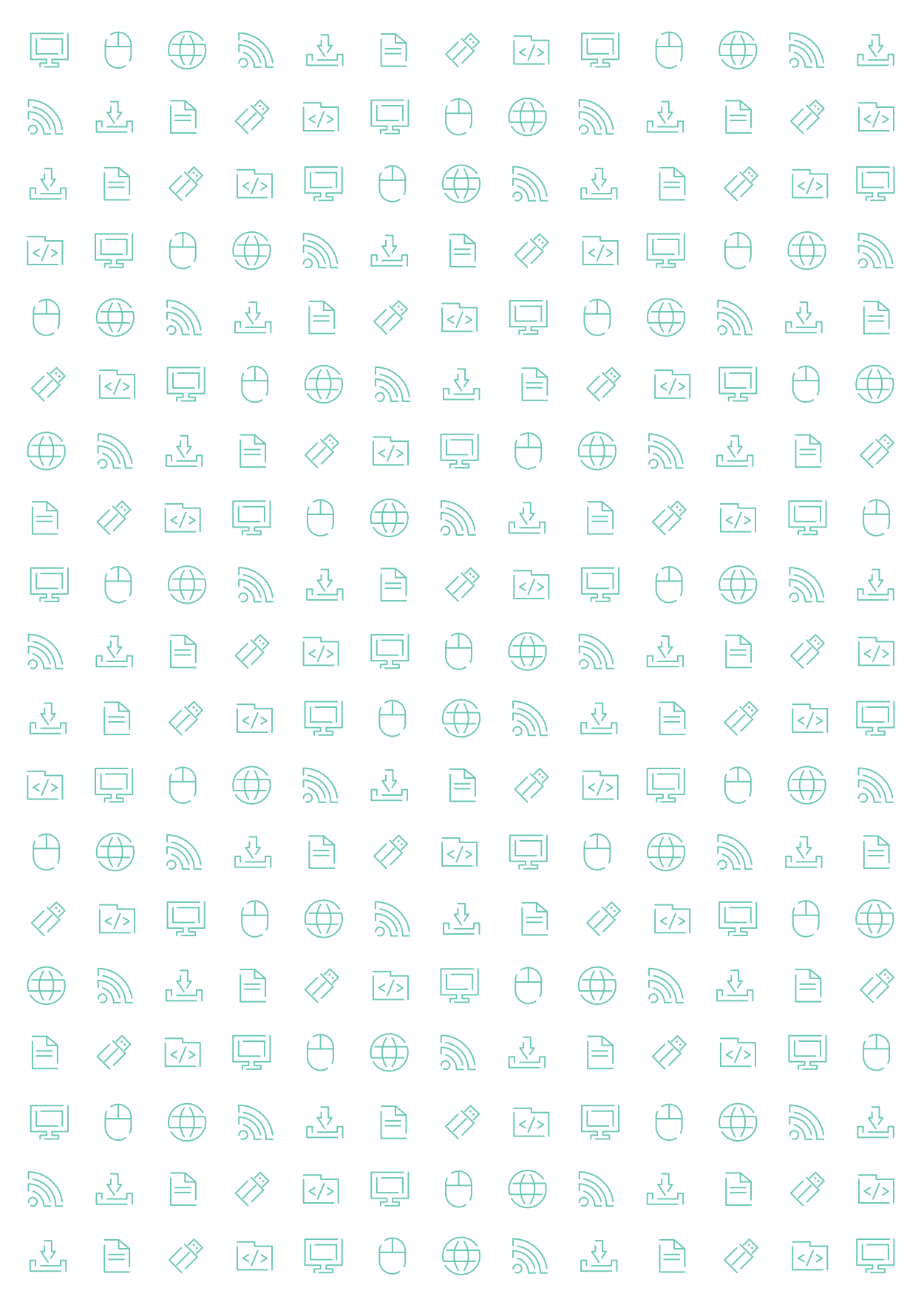
En general, cada vacuna se aplica en más de una dosis, por lo que se conoce de cada niño la fecha en que se administró cada vacuna y la dosis suministrada. A cada niño se le aplican diferentes vacunas y una vacuna se administra a varios niños. De cada niño y vacuna se poseen ciertas observaciones, por ejemplo, reacciones, contraindicaciones, etcétera. Crea la base de datos que permita controlar esta información. Utiliza al menos dos tablas en la base de datos y completa cada una con al menos cuatro registros.

Nota: en la tabla las palabras están juntas porque en SQLiteAdministrator, para completar los nombres de los campos, no se puede dejar espacio entre los caracteres.

5. Se desea controlar la actividad pioneril en una escuela primaria. De cada pionero se conoce su número de expediente, su nombre, su grado escolar y su sexo. Un pionero puede ser Moncadista o José Martí, se sabe también de cada Moncadista, si es o no seminterno y, por otro lado, de cada pionero José Martí se conoce la cantidad de veces que ha estado de vacaciones en un campamento de pioneros y la cantidad de horas de trabajo voluntario que ha realizado. Todos los pioneros participan en actividades. De cada actividad se conoce el número que la identifica, el tipo de actividad de que se trata, el lugar, la fecha y la hora en que se realizó. Un pionero participa en muchas actividades y en una actividad participan muchos pioneros.

Se conoce de cada pionero que participa en una actividad si llegó puntual o no. Para la mejor realización de cada actividad, se designa a un pionero José Martí como responsable de ella. Cada actividad tiene como responsable a un pionero José Martí y cada pionero José Martí puede ser responsable de solo una actividad. Además, para el mejor desarrollo del trabajo pioneril, cada pionero José Martí "apadrina" a varios pioneros Moncadista y un pionero Moncadista es "apadrinado" por un pionero José Martí.

a) Crea la base de datos que dé solución a esta problemática. Completa cada tabla con al menos cuatro registros y utiliza el criterio que desees para realizar al menos una consulta a alguna de las tablas creadas por ti.



CAPÍTULO 2

Profundizando en la programación

La programación es un campo de estudio que se enfoca en la creación de ***software*** y aplicaciones. Dentro de este campo, se encuentra la programación orientada a objetos que se basa en la definición y creación de objetos complejos que contienen tanto datos como métodos; estos objetos interactúan entre sí para construir aplicaciones complejas. Por otro lado, está la programación dirigida por eventos, un paradigma de programación en el que el flujo del programa está determinado por eventos o mensajes desde otros programas o hilos de ejecución. Es por ello que en la programación orientada a objetos y dirigida por eventos existen dos paradigmas de programación que se utilizan ampliamente en el desarrollo de aplicaciones modernas, donde el flujo del programa está determinado por eventos o mensajes y permite crear aplicaciones robustas, escalables, eficientes y fáciles de mantener.

Por otra parte, es importante destacar que estamos viviendo una época de profundos cambios, un mundo en el cual la tecnología es transversal a todos los ámbitos de la vida y nos desafía a desarrollar soluciones innovadoras a los problemas de nuestra vida cotidiana, al permitirnos aprender del trabajo colaborativo con personas que representan diversas culturas, religiones y estilos de vida, con un espíritu de respeto mutuo y de diálogo abierto en diferentes contextos.

Estas profundas transformaciones que impone la revolución de las Tecnologías de la Información y la Comunicación van abarcar diferentes ámbitos y aspectos, es decir, van a permear desde la comunicación, la producción de conocimiento, la circulación de la información, el vínculo de los gobiernos con los ciudadanos, el mundo del trabajo y la cultura, pasando por las relaciones cotidianas entre las personas, hasta esta nueva era cognitiva

donde los objetos y las personas pasan a estar, por defecto, conectadas (el denominado Internet de las cosas) y donde se verá la fusión de las inteligencias de las máquinas con las humanas.

Por lo antes expuesto, es que se hace necesario el desarrollo del pensamiento computacional, sobre todo, por las posibilidades que brinda en el desarrollo del pensamiento lógico, la resolución de problemas y su relación con la robótica y la inteligencia artificial, conjuntamente con las mejoras en la comunicación y el intercambio de experiencias.

¿Qué vas a aprender?

Recordaremos las ideas esenciales del pensamiento computacional, como metodología para la resolución de problemas; de la lógica de programación, de la robótica e inteligencia artificial, y el uso de LiveCode como aplicación libre y multiplataforma para la programación orientada a objeto y dirigida por eventos.

¿Para qué me sirve?

Para desarrollar proyectos de diverso tipo y resolver problemas mediante un lenguaje de programación de alto nivel.

¿Qué debo saber?

¿Qué es el pensamiento computacional, como metodología para la solución de problemas? Concepto de algoritmo e implementación de algoritmos con el entorno de programación LiveCode. ¿Qué es LiveCode? ¿Qué ventajas posee el trabajo con LiveCode? ¿Cuáles son los conceptos principales de LiveCode?

2.1 El pensamiento computacional y la lógica de la programación

Como recordarás, el denominado ***pensamiento computacional*** o ***algorítmico*** supone un modo de abordar y resolver problemas, sean estos del tipo que sean, ya que este tipo de pensamiento ayuda a la toma de decisiones de una manera ordenada, secuenciada y lógica, sin ambigüedades. Además, se encuentra muy relacionado a la programación, donde tienes la posibilidad de pasar de un simple usuario a un creador de nuevas soluciones.

Aunque existen diversas definiciones de pensamiento computacional, se plantea que este es un proceso para la solución de problemas a través de diferentes técnicas como: organización y análisis de información de manera lógica; representación de información mediante abstracciones, tales como modelaciones, simulaciones o diagramas; formulación de problemas que te permitan usar una computadora y otras herramientas para resolverlos; automatización de soluciones mediante el pensamiento algorítmico (una serie ordenada de pasos); identificación, análisis e implementación de posibles soluciones con el objetivo de lograr la combinación más efectiva y eficiente de pasos y recursos; así como la generalización y transferencia de este proceso de resolución a una amplia variedad de problemas.

Recordemos ahora algunas de las técnicas que se emplean como parte del pensamiento computacional, como metodología para resolver problemas cotidianos, diseñar sistemas domésticos y realizar tareas rutinarias.

Metodología general

En la ***Metodología general*** la solución de problemas pasa por cuatro etapas principales, que son las siguientes:

1. Etapa de recolección de información: entender el problema es esencial para resolverlo, por lo que se requiere hacer una investigación extensa acerca de este: qué es lo que se desea obtener, qué se necesita, qué obstáculos se pueden presentar. Un factor clave para esta fase es la habilidad de discriminación de la información para diferenciar qué es relevante y qué no lo es, en la que interviene un proceso de ***abstracción.***
2. Etapa de generación de soluciones: en circunstancias normales, a mayor cantidad de posibles respuestas hay una mayor oportunidad de resolver un contratiempo, dado que de esta manera tenemos más alternativas para escoger. Una estrategia muy utilizada en este período es la de ***ensayo y error,*** en la que se intenta una solución y, si no funciona, se prueba una segunda hasta resolver la situación.
3. Etapa de implementación de la solución: esta comienza con la toma de decisiones, y una vez que se han propuesto algunas alternativas, es necesario llevar a cabo una de ellas. En algunas situaciones, el problema se puede resolver de más de una forma; sin embargo, algunas soluciones pueden ser mejores que otras porque llevan más o menos tiempo o pueden ser más fáciles o eficientes al momento de aplicarse. Esta fase implica llevar a cabo un plan de acción. Para mucha gente, es difícil

seguirlo, especialmente con problemas complejos, por lo que el compromiso de los involucrados es un punto clave del proceso, si no existen estos, entonces las intenciones y el plan de acción no significan nada.

4. Etapa de evaluación: es el último peldaño de esta metodología. Una vez que la solución o plan de acción ha sido implementado, se necesita considerar si el objetivo principal se ha cumplido, si no es así, es necesario considerar otra alternativa o plan de acción. Para esto se debe revisar cada acción y, si es necesario, retomar la etapa dos: generación de soluciones.

Metodología de Polya

Como conoces de grados anteriores, George Polya en su libro ***Cómo resolverlo*** (1945), propone un método de cuatro pasos para la resolución de problemas matemáticos. Estos pasos se exponen a continuación:

1. Entender el problema.
2. Definir o diseñar un plan.
3. Llevar a cabo el plan.
4. Mirar atrás.

1. Entender el problema: para esto debes atender las siguientes interrogantes: ¿puedes describir el problema en tus propias palabras?, ¿qué tratas de hacer o encontrar?, ¿qué es lo que se desconoce?, ¿qué información obtienes del problema?, ¿qué información falta o no se necesita del todo?
2. Definir o diseñar un plan: busca un patrón, examina situaciones relacionadas y determina si la misma técnica puede ser aplicada; examina un caso especial o más simple para comprender el problema original, haz una tabla, haz un diagrama; escribe una ecuación, usa prueba y error, trabaja hacia atrás e identifica una submeta.
3. Llevar a cabo el plan: implementa la estrategia o estrategias planteadas en el paso 2, y ejecuta cualquier operación o acción necesaria, revisa cada paso teniendo en cuenta cómo vas avanzando, esto puede ser intuitivo o de manera formal, y lleva un registro exacto de tu trabajo.
4. Mirar atrás: revisa los resultados del problema original y, de ser necesario, consigue una prueba e interpreta la solución en estos términos: ¿tu respuesta es coherente y razonable?; determina si hay manera de encontrar otra y, si es posible, determina problemas más generales o relacionados en los que puedas aplicar esta técnica.

Como recordarás estas metodologías te servirán para resolver cualquier tipo de problema, para luego analizar, si es posible, su implementación en un sistema informático y obtener como resultado final un programa o aplicación informática.

Ahora te propongo que utilices las técnicas que componen la metodología del pensamiento computacional para resolver los siguientes problemas:

1. En una fábrica se elaboran N piezas y de ellas se consideran buenas aquellas cuyo diámetro oscila entre 20 cm y 23 cm. Determina el promedio del diámetro de las piezas buenas y el por ciento de las piezas en mal estado. ¿Cuál fue el razonamiento que seguiste? ¿Cuáles fueron las entradas, salidas y el problema por resolver?

2. Juan Felipe es jefe de bodega en una fábrica de pañales desechables y una de las tareas del día consiste en llamar al proveedor de los empaques y ordenarle la cantidad suficiente de cajas para empacar los pañales fabricados en la semana próxima. El jefe de producción le informó ayer a Juan Felipe que la producción diaria será de 744 pañales y en cada caja cabe una docena de ellos. ¿Qué debe hacer Felipe? ¿Cuál fue el razonamiento que seguiste? ¿Cuáles fueron las entradas, salidas y el problema por resolver?

3. Utiliza la metodología de George Polya para resolver los siguientes ejercicios. Para realizar el análisis de estos, es importante que incluyas los siguientes pasos en cada situación planteada:
   a) Entiende el problema (descríbelo con tus propias palabras).
   b) Sugiere una solución (haz un diagrama, emplea prueba y error, plantea una ecuación, reconoce patrones, etcétera).
   c) Implementa la solución.
   d) Mira hacia atrás y reflexiona si tu respuesta es correcta, si puedes hacerlo de otra manera o si existen otras soluciones más viables.

   - En la preparación de un atleta que compite en 3000 m con obstáculos, este se entrena corriendo diariamente. Se tiene el control de los kilómetros recorridos cada día de una etapa de su entrenamiento, y se desea conocer el total de kilómetros recorridos, el promedio de kilómetros recorridos por vía y el número de días en que no corrió.

- Se tiene la producción de cada día de una semana de un central azucarero y se desea conocer la cantidad de veces que cumplió la norma potencial de 200 000 arrobas, el total producido en la semana y el promedio de molienda por día.

Características generales de los programas

Como estudiaste en grados anteriores, los ***programas*** están constituidos por un conjunto de sentencias que se procesan en una determinada secuencia y conforman órdenes capaces de manipular una serie de datos con el fin de obtener un determinado resultado. Las órdenes o instrucciones pueden dividirse en tres grandes secciones, cada una de las cuales corresponde a una parte de la codificación del programa:

- Entrada de datos: aquí se engloban todas aquellas instrucciones que recogen datos de un dispositivo o periférico (por ejemplo, el teclado), que se almacenan después en la memoria central o principal para su proceso posterior.
- Proceso del algoritmo: en esta parte del programa se describen las instrucciones o sentencias encargadas de procesar los datos recogidos en la sección anterior, conforme al propósito o la finalidad del programa. Los resultados obtenidos se almacenan nuevamente en la memoria principal.
- Salida de datos o resultados: este bloque está formado por las instrucciones que toman los resultados obtenidos en la etapa anterior, que se envían a los dispositivos de salida de la información (por ejemplo, la pantalla).

El desarrollo de un programa requiere los siguientes pasos:

- Definición y análisis del problema.
- Diseño del algoritmo.

Recordemos que un ***algoritmo*** es una secuencia de pasos precisos que conducen a la solución de un problema. Estos deben cumplir con las siguientes reglas o propiedades:

- Deben ser finitos, o sea, deben tener una cantidad bien definida de pasos.
- Los pasos deben estar ordenados.
- Los pasos deben ser claros e igualmente comprensibles para cualquier persona (libres de ambigüedad).
- Codificación del programa (obtendremos el código fuente).
- Compilación (obtendremos el código objeto).

- Depuración de errores y verificación del programa.
- Explotación (documentación y mantenimiento).

Los ***datos*** de un programa son aquellos elementos que constituyen unidades de tratamiento de la información que se tiene que procesar. Para que esta información sea correctamente manipulada, los programas deben definir las estructuras de datos conforme a un identificador, un tipo y un valor.

El ***identificador*** es el nombre que le damos al dato dentro del programa para poder hacer referencia a él. El ***tipo*** establece la naturaleza y el rango (intervalo) de valores que puede almacenar. Finalmente, el ***valor*** es el contenido del dato respecto al tipo definido.

La utilización de los datos depende del lenguaje de programación que se utilice para desarrollar la aplicación y del tipo de dato adecuado para almacenar determinada característica (tabla 2.1). En la mayoría de los lenguajes de programación hay tipos de datos primitivos, a partir de los cuales se definen las variables que se van a utilizar. Los ***datos primitivos*** se clasifican usualmente en: numéricos, lógicos y de carácter.

Tabla 2.1 Tipos de datos más usados

Tipos de datos	Valores
Numéricos (enteros, reales, etcétera)	Magnitudes numéricas.
Booleanos	Solo admite dos valores (verdadero/falso o sí/no).
Caracteres o cadenas (strings)	Conjunto de caracteres reconocidos por la PC.
Punteros	Contiene la dirección de memoria de otra variable.
Tablas (arrays)	Estructuras compuestas por filas y columnas.
Listas, pilas o colas	Elementos lineales enlazados.

Tipos de datos	Valores
Árboles o grafos	Elementos no lineales enlazados.
Ficheros de bases de datos	Archivos compuestos por registros.

Así mismo los datos pueden ser ***constantes,*** cuyo valor no cambia a lo largo del programa, o ***variables,*** cuyo contenido puede ir cambiando a lo largo de la ejecución.

En el caso de las variables, recuerda que inicializar una variable es asignarle de manera explícita un valor inicial al espacio de memoria que la variable va a ocupar. Esta operación es imprescindible si la variable está sujeta a cambios como una consecuencia de operar con su contenido.

Las variables pueden ser inicializadas al principio del programa o de su procedimiento, o bien pueden permanecer sin recibir un valor previo hasta que en una instrucción se lo asigne. La mayoría de los lenguajes de programación exigen definir las variables al principio, de modo que estas primeras órdenes constituyen instrucciones de definición de datos.

Es importante tener en cuenta que cada una de las variables y constantes debe tener un nombre que las identifique de forma única. Dependiendo del lenguaje de programación que se emplee, las reglas de nomenclatura varían; sin embargo, estas deben cumplir con los siguientes lineamientos:

1. Solo contienen caracteres alfabéticos, números y guion bajo.
2. Comienzan con un carácter alfabético.
3. Son nemotécnicas, o sea, significativas, es decir, están relacionadas con el valor que guardan. Por ejemplo, la variable "peso" indica que el contenido es el peso de algo o alguien.
4. No se extienden a más de 255 caracteres, aunque esto puede variar de acuerdo con el lenguaje de programación con que se trabaje.

Recuerda que el único alfabeto que comprenden los sistemas informáticos son los códigos binarios, o sea, 0 (ceros) y 1 (unos), de ahí que, increíblemente y por suerte hasta el momento, a pesar de los avances de la tecnología, hoy en día los sistemas informáticos solo pueden procesar

estos tres tipos de datos y el resto de atributos de las cosas se reducen a estos mismos tipos:

- Datos numéricos
- Datos de cadenas o alfanuméricos
- Datos Lógicos o Boolean (Verdadero o Falso)

Inclusive los datos audiovisuales y físicos se reducen a estos tipos de datos.

Recordemos el significado de las siguientes expresiones:

N = 8 (a un espacio de memoria, al que se le ha llamado ***N,*** se le asigna o deposita el valor 8)
Var= "B" (***B*** un espacio de memoria, al que se le ha llamado ***Var,*** se le asigna o deposita el valor de cadena ***B.***)
X = X + 1 (como debes estar pensando esta expresión desde el punto de vista matemático es absurda, ya que, bajo el supuesto de que ***X*** es un número entero, la expresión plantea que el número ***X*** es igual a su sucesor ***X + 1,*** y como sabemos esto nos llevaría al absurdo de que ***0=1,*** algo así como que el todo es igual a la nada)

Sin embargo, el hecho de que la informática tenga tanta relación con la matemática hizo que un mismo signo (=) se aplicara en ambas ciencias de manera distinta. En matemática el signo (=), significa ***igualdad,*** mientras que en informática el signo (=) significa ***asignación. Igualdad*** pretende decir, que lo que está a la izquierda es igual a lo que está a la derecha y ***asignación*** significa que lo que está a la derecha deberá reemplazar el valor que está a la izquierda, por tal motivo la expresión informática ***X*** = ***X*** + ***1*** quiere decir que donde estaba ***X*** se cambie por el valor que estaba, o sea, el valor de ***X***, adicionándole 1. ¿Interesante verdad?

Tan delicado es el asunto que algunos lenguajes de programación han abolido el ***signo de igualdad*** como equivalente del ***signo de asignación.*** Por consiguiente, al escribir correctamente la expresión ***X***= ***X***+***1***, escribiremos ***X:***= ***X + 1***. Esta expresión representa un ***contador.***

Recuerda que un ***contador*** en informática se entiende como una expresión del tipo: ***C:= C + 1***, y son variables que cuentan el número de eventos ejecutados dentro de un algoritmo. Este aumenta o disminuye su contenido en un valor constante. El contador inicia generalmente en 0 o 1.

Las operaciones básicas que se llevan a cabo con una ***variable contador*** son las siguientes:

1. Inicialización: ***contador*** = ***valor inicial.*** Esta operación asigna el valor inicial de la variable contador. Por ejemplo: ***vidas*** = 1.
2. Incremento o decremento:
   Para el incremento: ***contador*** = ***contador*** + ***constante,***
   y para el decremento: ***contador*** = ***contador*** – ***constante.***

Dichas operaciones se realizan cada vez que ocurre el evento que se desea contar. Usualmente el incremento o decremento corresponde a la constante, por ejemplo: ***vidas*** = ***vidas*** + 1. Desde luego, pueden utilizarse otras constantes, por ejemplo: ***total*** = ***total*** + 5.

De igual manera, si recuerdas lo estudiado en grados anteriores, un ***acumulador*** o ***sumador*** en informática se entiende como una expresión del tipo: ***S:***= ***S*** + ***k,*** en la que ***k*** es valor arbitrario.

Los acumuladores incrementan o decrementan su contenido en un valor variable. La acumulación inicia generalmente en 0 o 1. La operación básica que se debe realizar con ellos es la siguiente:

1. Inicialización: ***acumulador*** = ***valor inicial***. Esta operación asigna el valor inicial de la variable acumulador y depende del tipo de operación que se va a realizar; por ejemplo: ***energía*** = 100.
2. Incremento o decremento:
   Para el incremento: ***acumulador*** = ***acumulador*** + ***variable***,
   y para el decremento: ***acumulador*** = ***acumulador*** – ***variable***

Dichas operaciones se realizan cada vez que ocurre el evento que se desea acumular. Por ejemplo: ***vitalidad*** = ***vitalidad*** – ***impacto.***

Si hablamos de los operadores, recordemos que son símbolos que permiten conectar o relacionar los datos entre sí, con lo que se facilita la realización de las diversas operaciones. A continuación verás cuáles son los tipos de operadores que puedes utilizar:

- Operador de asignación (:=): si se desea que dentro de la variable ***Volumen*** se coloque el valor 5, se escribirá: ***Volumen***:= ***5***
- Operadores aritméticos: la jerarquía de los operadores es similar a lo que aprendiste en Matemática, primero se opera lo que está entre paréntesis y luego, la potencia, la multiplicación, la división, la división entera y el módulo, según aparezcan y finalmente, la suma y la resta (tabla 2.2).

Tabla 2.2

Operador	Operación	Ejemplo	Resultado
paréntesis	Agrupamiento	(2+3)*5	25
^	Potencia	4^4	16
*	Multiplicación	5,25*3	15,75
/	División	100/4	25
+	Suma	2+4	6
-	Resta	4-2	2
Mod	Módulo de la división entera	4 mod 3	1
Div	División entera	8 div 3	2
Sqrt	Raíz cuadrada	sqrt(16)	4
Abs	Valor absoluto	abs(-4)	4
random	Número aleatorio	random(10)	Un número al azar entre 1 y 10

- Operadores relacionales: son los operadores que sirven para comparar (tabla 2.3).

Tabla 2.3

Operador	Operación	Ejemplo	Resultado
=	Igual que	"Lalo"="Lola"	Falso
<>	Desigual a	"Lalo"<>"Lola"	Verdadero
<	Menor que	4 < 8	Verdadero

Operador	Operación	Ejemplo	Resultado
>	Mayor que	4 > 8	Falso
<=	Menor o igual	5 <= 12	Verdadero
>=	Mayor o igual	5 >= 12	Falso

- Operadores lógicos: son operadores que permiten conformar condicionales a partir de expresiones simples, y pueden ser de tres tipos:
  - Conjunción (y)
  - Disyunción (o)
  - Negación (No)

- Operadores alfanuméricos: es un operador de Concatenación:
  Cad1 ***Unir*** Cad2 **=** Cad1Cad2. Por ejemplo:
  María ***Unir*** Elena = MaríaElena

Recuerda que...

Los paréntesis permiten alterar el orden lógico de las operaciones:

Ejemplo: A/(6+8)

Según el orden de operaciones previsto se realiza siempre la división antes que la suma; sin embargo, al estar la suma entre paréntesis, debe realizarse la suma antes y luego la división.

Descripción de algoritmos

Recuerda que...

Los algoritmos pueden ser representados de cuatro maneras diferentes, mediante: el lenguaje natural, un diagrama de flujo, un pseudocódigo, y un lenguaje de programación o implementación.

Ahora podrás recordar la expresión de algoritmos mediante diagramas de flujo y de pseudocódigo, pero antes de iniciar el análisis y la construcción de algoritmos mediante diagramas de flujo, es significativo señalar

que estos no son solo importantes en informática, sino en todos los procesos que llevan consigo una secuencia lógica.

Los ***diagramas de flujo*** son la representación gráfica y visual de cada paso del algoritmo por medio de símbolos sencillos de fácil comprensión, que representan el flujo de los datos en la solución de un problema, así como las operaciones ejecutadas sobre los datos (figura 2.1).

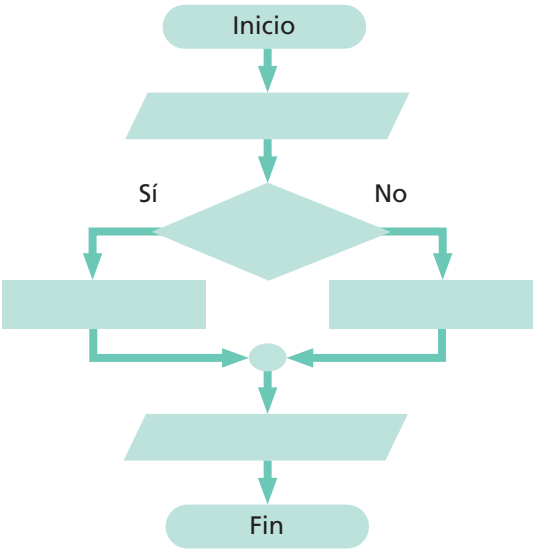


Fig. 2.1

Hay símbolos aceptados como estándar, a partir de las propuestas de organizaciones como: American National Standards Institute (ANSI) y la International Organization for Standarization (ISO) (figura 2.2).

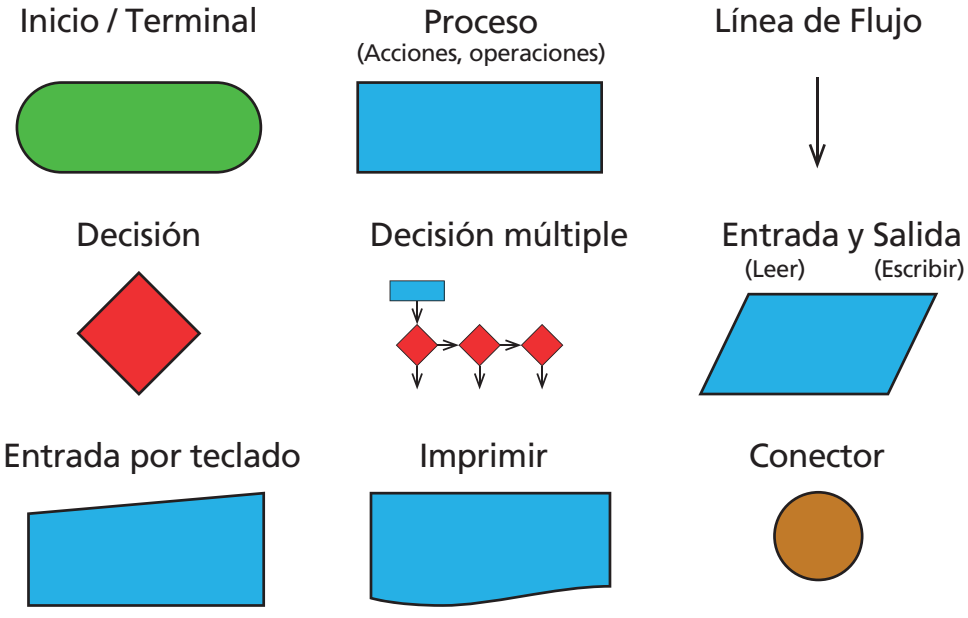


Fig. 2.2

Por su parte, la expresión de ***algoritmos mediante pseudocódigo*** no es más que la formalización o elección de un subconjunto de palabras del lenguaje natural mezclado con algunos símbolos para expresar algoritmos.

A continuación, se proponen algunos ejemplos en los que se expresan ambos tipos de algoritmos (el de diagrama de flujo y el de pseudocódigo).

Ejemplo

El policlínico Cristobal Labra del municipio La Lisa tiene varios puntos de reparto de la vacuna Abdala que se utilizan en la inmunización de la población cubana contra la Covid-19, que se pretende funcionen de la siguiente manera: cada día, empezar con 1 000 vacunas disponibles y a través de un programa que controla las entregas se necesita avisar si el inventario baja de 200 unidades.

a) Desarrolla un algoritmo de pseudocódigo y otro de diagrama de flujo que represente este algoritmo de trabajo.

- Pseudocódigo:

1. ***Inicio [Control de Vacunas aprenderaprogramar.com]***
2. Existencias = 1000
3. ***Mientras Existencias >= 200 Hacer***
   Mostrar "Introduzca el número de unidades entregadas"
   Pedir Entregadas
   Existencias = Existencias – Entregadas

Repetir

4. Mostrar "El inventario ha bajado de 200 unidades.
Debe comunicarlo"
5. ***Fin***

Si te das cuenta, al analizar el pseudocódigo la variable ***Existencias*** funciona como un acumulador que parte de un valor inicial, y este valor, tras un movimiento, depende de su contenido precedente.

- Diagrama de flujo (figura 2.3):

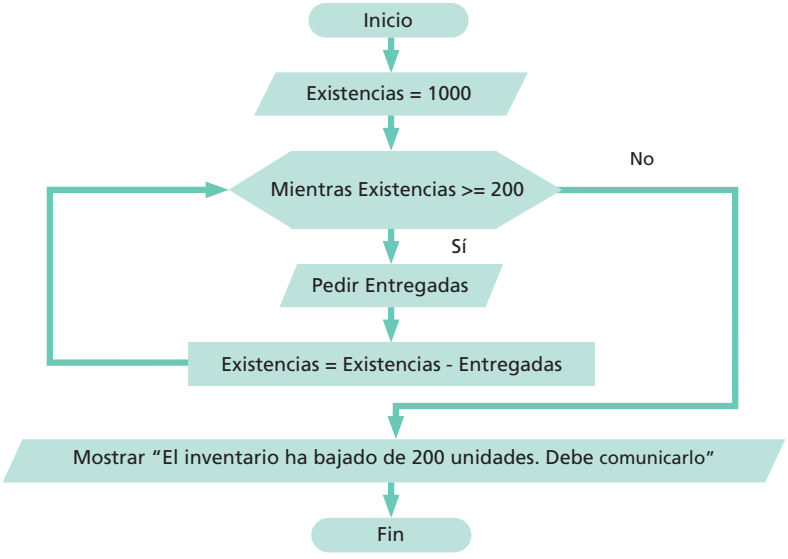


Fig. 2.3

2.2 La programación estructurada

Los ***lenguajes estructurados*** son aquellos en los que las sentencias incluidas en el código se ejecutan de forma lineal y sencilla (una detrás de otra) a partir de un solo punto de entrada (inicio) y otro de salida (final). Recordemos ahora el teorema fundamental de la programación estructurada.

Recuerda que...

Teorema fundamental de la programación estructurada: establece que toda función computable pueda ser implementada en un lenguaje de programación que combine solo tres estructuras lógicas o de control de flujo. Estas tres formas son:

Secuencial: son un grupo de instrucciones sucesivas que se ejecutan de forma ordenada y seguida (figura 2.4).

Alternativa, condicional o selectiva: son instrucciones que permiten establecer condiciones. En función de si estas se cumplen o no, se ejecutan unas instrucciones u otras (si <condición>entonces <instrucción 1>en caso contrario qq1<instrucción 2>; o, según la terminología inglesa: if <condición>then <instrucción 1> else <instrucción 2>). Las estructuras condicionales pueden ser simples o múltiples, en función de las respuestas que pueda tener la condición (figuras 2.5 y 2.6).

Cíclica, repetitiva o de iteración: son instrucciones que se repiten en un número limitado de veces o hasta que se cumpla una determinada condición (mientras <condición> haz <instrucción>; en ingles ***while*** <condición> ***do*** <instrucción>). (figura 2.7).

Estructura secuencial

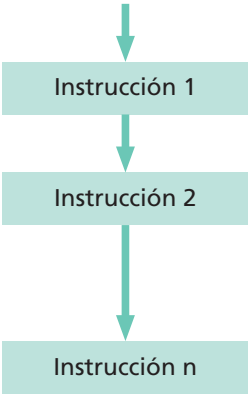


Fig. 2.4

Estructura condicional simple

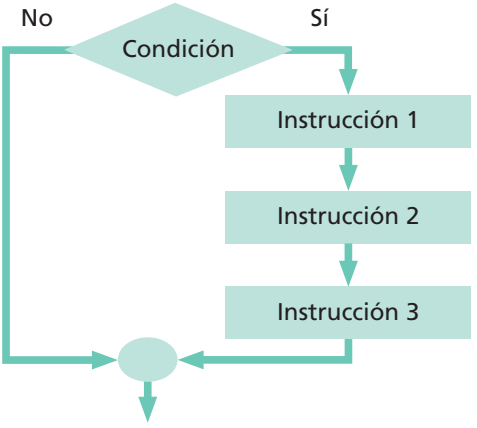


Fig. 2.5

Estructura condicional múltiple

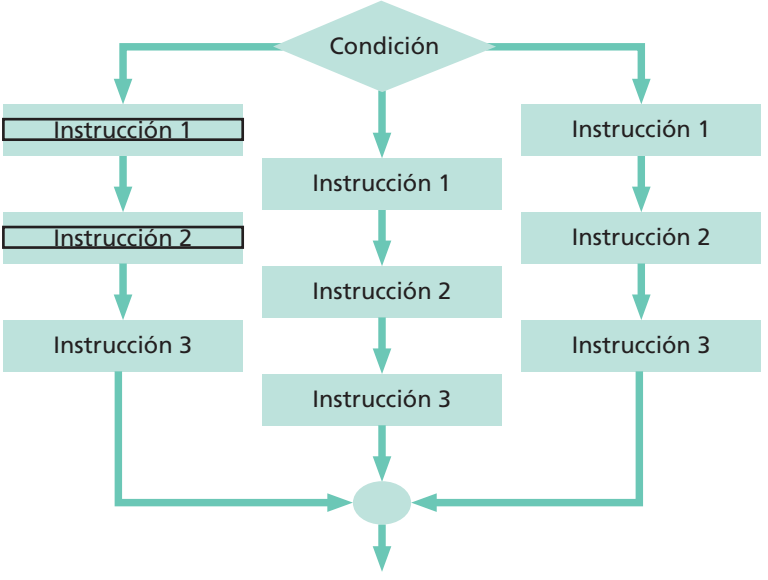


Fig. 2.6

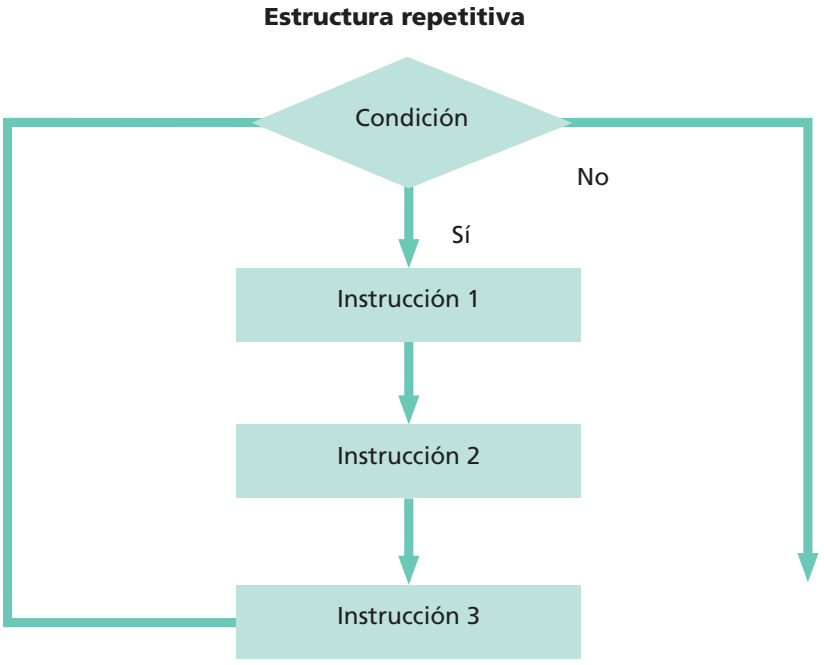


Fig. 2.7

A continuación se proponen tres ejemplos de cómo se utiliza el diagrama de flujo (figura 2.8) y el de pseudocódigo, para representar algoritmos que utilicen las diferentes estructuras de control de flujo.

Ejemplo 1

Elabora un algoritmo para calcular el área de cualquier triángulo rectángulo y presenta el resultado en pantalla.

- Pseudocódigo:
  Paso 1: Inicio
  Paso 2: Asignar el número 2 a la constante "Div"
  Paso 3: Conocer la base del triángulo y guardarla en la variable "Base"
  Paso 4: Conocer la altura del triángulo y guardarla en la variable "Altura"
  Paso 5: Guardar en la variable "Área" el valor de multiplicar "Base" por "Altura"
  Paso 6: Guardar en la variable "Área" el valor de dividir "Área" entre "Div"
  Paso 7: Reportar el valor de la variable "Área"
  Paso 8: Final

- Diagrama de flujo (figura 2.8):

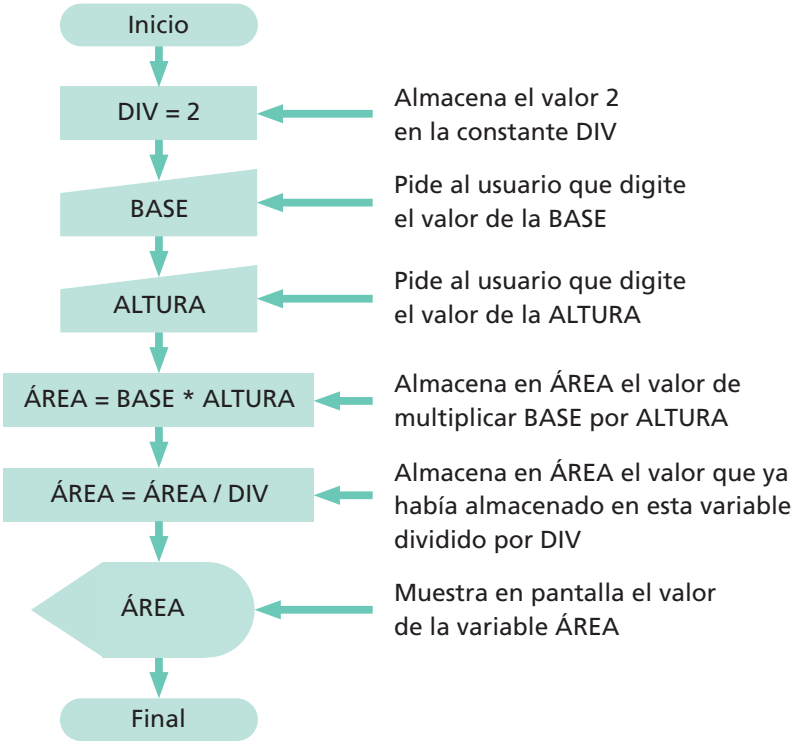


Fig. 2.8

Ejemplo 2

Desarrolla un algoritmo que permita leer dos valores distintos, determina cuál de los dos valores es el mayor y escríbelo.

- Pseudocódigo:
  1. ***Inicio***
  2. Inicializar variables: A = 0, B = 0
  3. Solicitar la introducción de dos valores distintos
  4. ***Leer*** los dos valores
  5. Asignarlos a las variables A y B
  6. ***Si*** A = B ***Entonces*** vuelve a 3 porque los valores deben ser distintos
  7. ***Si*** A>B ***Entonces Escribir*** A, "Es el mayor"
  8. ***De lo contrario: Escribir*** B, "Es el mayor"
  9. ***Fin_Si***
  10. ***Fin***

- Diagrama de flujo (figura 2.9):

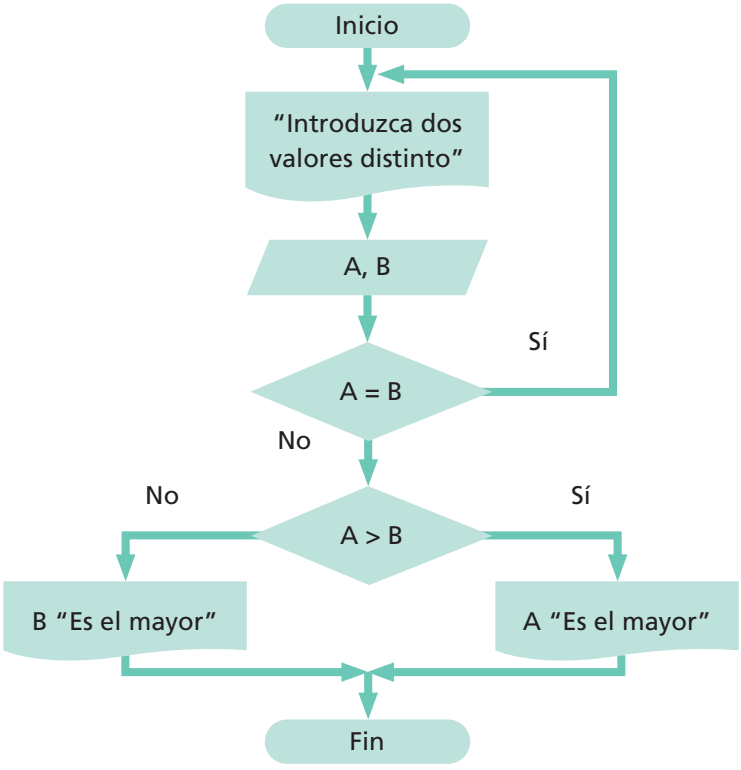


Fig. 2.9

Ejemplo 3

Desarrolla un algoritmo que permita leer un valor cualquiera N y escribe si dicho número es par o impar.

- Pseudocódigo:
  1. Inicio
  2. Declaración de variables: N
  3. Leer un número
  4. Asignarlo a la variable N
  5. ***Si*** el residuo de dividir a N entre 2 es igual a cero
  6. **Si** es Si: ***Entonces:*** Escribir "Es par"
  7. ***Sino:*** Escribir "Es impar"
  8. ***Fin_Si***
  9. Fin

- Diagrama de flujo (figura 2.10):

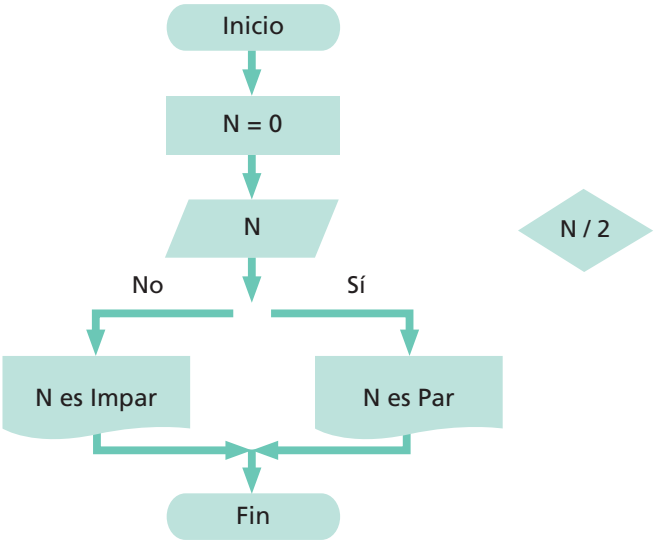


Fig. 2.10

Ahora se te propone desarrollar las siguientes actividades:

1. Elabora un algoritmo en pseudocódigo para cada uno de los siguientes problemas y luego conviértelos en un diagrama de flujo:

a) Halla el perímetro de un cuadrado cuyo lado mide 5 cm.

b) Halla el área de un cuadrado cuyo lado mide 5 cm.

c) Halla uno de los lados de un rectángulo cuya área es de 15 cm^2 y uno de sus lados mide 3 cm.

d) Halla el área y el perímetro de un círculo cuyo radio mide 2 cm.

e) Halla el área de un pentágono regular de 6 cm de lado y 4 cm de apotema.

f) Dada la concentración de una disolución, determina si es básica, ácida o neutra. Recuerda que: pH = - lg C
Si pH >7, es base
Si pH < 7, ácida
Si pH = 0, neutra

g) Dadas las notas (nt) de los N educandos de nombre (nb) de un grupo, se deberá conocer:
- La condición de cada uno en la asignatura analizada.
- La cantidad de educandos suspensos.
- El promedio de notas del grupo en la asignatura.

2. Dada la descripción algorítmica propuesta, diga:

a) ¿Qué problema resuelve?

b) Si dispusieras de la operación de elevar al cubo directamente un valor cualquiera, ¿cómo harías la descripción algorítmica de este problema?

1. Datos: a, b, x
2. p = a * x
3. q = p * x
4. r = q * x
5. y = r + b
6. Imprimir: y
7. Fin

3. Analiza si las descripciones algorítmicas siguientes son dos descomposiciones de un mismo algoritmo o no:

1) Datos: b, x, y
2) $a_1 = b * y$
3) $a_2 = a_1 + x$
4) $a_3 = a_2 * y$
5) Imprimir: a_3
6) Fin

1) Datos: b, x, y
2) $a_1 = b * y$
3) $a_2 = a_1 * y$
4) $a_3 = x * y$
5) $a_4 = a_2 + a_3$
6) Imprimir: a_4
7) Fin

a) ¿Qué problema resuelve?

b) Realiza el diagrama de flujo para cada uno.

2.3 La programación orientada a objetos y conducida por eventos

En este epígrafe vamos a recordar que los programas informáticos se escriben mediante lo que se denomina ***Lenguaje de programación,*** y que de este tipo de lenguaje existen cientos.

En principio, la elección de un lenguaje de programación en particular obedece a múltiples criterios, desde los que se ponen de moda, hasta los que resultan demasiado difíciles de aprender por poseer una sintaxis relativamente abstracta y críptica. También es cierto que todos los seres humanos estamos capacitados para leer y escribir en una lengua natural, por tal motivo, en la medida que el lenguaje de programación esté

más cercano a un lenguaje natural, este, sin dudas, nos deberá resultar más asequible.

Los lenguajes orientados a objetos son la evolución lógica de la programación estructurada. La ***programación orientada a objetos*** (POO u OOP, según sus siglas en inglés) se basa en dividir los programas en pequeñas unidades lógicas de código. A estas pequeñas unidades lógicas de código se les llama ***objetos.*** Los ***objetos*** son unidades independientes que se comunican entre ellos mediante ***mensajes,*** los cuales poseen una serie de características especiales y pueden ser reutilizados.

Los objetos

La POO intenta describir de forma abstracta la forma de pensar de los seres humanos. Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo, diríamos que un auto es un objeto que tiene una serie de características (como podrían ser la marca, el modelo y el color) y una serie de funcionalidades asociadas (como ponerse en marcha o frenar). En el esquema de la POO, el coche sería el ***objeto;*** las ***propiedades o atributos*** serían las características (color, modelo, etcétera); y los ***métodos*** serían las funcionalidades asociadas (ponerse en marcha, frenar, etcétera). Un objeto de ***software*** mantiene sus características en una o más ***variables*** e implementa su comportamiento con métodos. Un ***método*** es una función o ***subrutina*** (pequeño programa) asociada a un objeto.

Las clases

En el mundo real, normalmente tenemos muchos objetos del mismo tipo. Por ejemplo, hay muchas marcas de autos. Si hablamos en términos de la POO, podemos decir que nuestro objeto auto es una instancia de una clase conocida como automóvil. Todos los autos tienen unas características (marca, modelo, color, motor, ventanas, ruedas, etcétera) y unos comportamientos (ponerse en marcha, frenar, acelerar, parar, ir marcha atrás, etcétera). Cuando los fabricantes de automóviles fabrican coches, aprovechan el hecho de que todos los autos deben compartir sus características comunes y elaboran modelos o plantillas que luego aplican a todos los autos de ese mismo modelo. Esos modelos o plantillas son lo que se conocen como ***clases,*** y a cada auto que se fabrica utilizando esos modelos se le denomina ***objeto.***

La herencia

Imagina ahora que el fabricante de automóviles, partiendo del modelo original, realiza dos modelos de auto distintos y a cada uno de ellos le otorga una característica diferente, realizando un modelo para autos con cambio manual y otro para autos con cambio de marchas automático. Cada modelo de automóvil hereda de su predecesor todos los atributos que este tenía (marca, modelo, color…) y sus propiedades (ponerse en marcha, frenar, acelerar…), y además cada uno incorpora un nuevo atributo (cambio). El primer modelo (clase automóvil) se convertirá en una ***superclase,*** ya que de ella se han derivado otras, y las clases generadas recibirán el nombre de ***subclases.***

Envío de mensajes

Los objetos por sí mismos no tienen mucha utilidad. Por ejemplo, un auto, sin una persona que lo conduzca, pierde gran parte de su significado. Para relacionar distintos objetos se emplean los ***mensajes.*** Así, una persona consigue que un coche gire a la derecha moviendo el volante hacia ese lado. Para relacionar objetos se utilizan ***métodos,*** con los cuales se pueden incluir ciertos ***parámetros*** o ***variables.*** Por ejemplo, al objeto auto se le podría enviar un mensaje con el método ***girar*** ( ) indicándole como parámetro la dirección (derecha o izquierda). También se puede invocar a métodos sin necesidad de pasar ningún parámetro, como se haría en el caso de los métodos ***frenar*** ( ) y ***acelerar*** ( ). En términos generales, se podría decir que los ***mensajes*** son la forma de realizar acciones sobre los objetos.

Algunas características generales de LiveCode y su entorno de trabajo

Después de este recordatorio necesario sobre algunos conceptos de suma importancia de la programación orientada a objetos, pasaremos a recordar los elementos fundamentales de una herramienta muy moderna, poderosa y profesional que te ha servido de base para ulteriores desempeños en esta disciplina, sobre todo si se tiene en cuenta el estado actual de la informática vinculada con el auge de los dispositivos móviles. Seguro que ya sabes que se está hablando de ***LiveCode,*** un lenguaje de programación que estudiaste en décimo grado y que ahora será analizado de manera breve, para luego adentrarnos en el mundo de la ingeniería del ***software.***

Como estudiastes en el grado precedente, el ***LiveCode*** es un ambiente de programación basado en objetos y dirigido por eventos con un lenguaje muy sencillo, ya que es casi como hablar en inglés.

Es un lenguaje que tiene aproximadamente ***2 600*** (un poco más) sentencias de programación y estas se pueden extender mediante "***plugins***" o programándolas directamente en lenguaje "***C***".

La filosofía o ***metáfora*** de esta herramienta de desarrollo se basa en una "***pila de tarjetas*** o una ***baraja de cartas,*** cada una de las cuales constituye ***una ventana*** de la aplicación. Existe una ***pila principal*** denominada "Main Stack" que contiene ***Cards*** o ***tarjetas*** (figura 2.11).

Fig. 2.11 Pila Principal o MainStack

La pila principal puede a su vez estar acompañada por otras pilas, cada una con su sistema de tarjetas (figura 2.12).

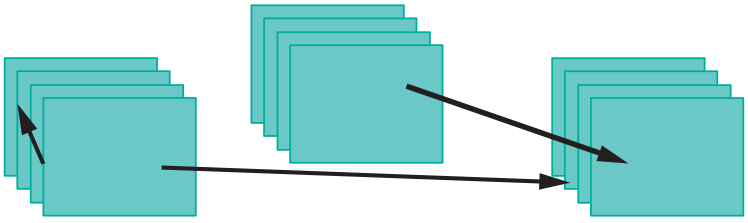


Fig. 2.12

El desarrollo de una aplicación puede concebir la navegación entre tarjetas de diferentes pilas.

¿Cómo se desarrolla una aplicación LiveCode?

Para la ejecución de la aplicación LiveCode se siguen los pasos siguientes:

1. Cuando se inicia la herramienta de desarrollo, ya sea a partir del icono de acceso directo o por su inicio en el menú de los programas, se tiene que crear un nuevo proyecto.

   El nuevo proyecto se puede crear dando clic en la opción ***New (Nuevo)*** del ***Start Center (Centro de Inicio),*** como se muestra en la figura 2.13, o seleccionando en el menú ***File (Archivo)*** la opción ***New Stack (Nueva pila)*** y seguido se optará por la variante ***Default Size (Tamaño implícito)***, para luego adaptarla a nuestras necesidades (figura 2.14).

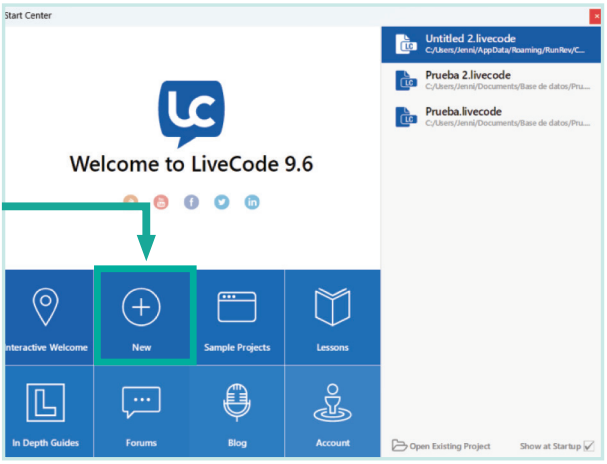


Fig. 2.13

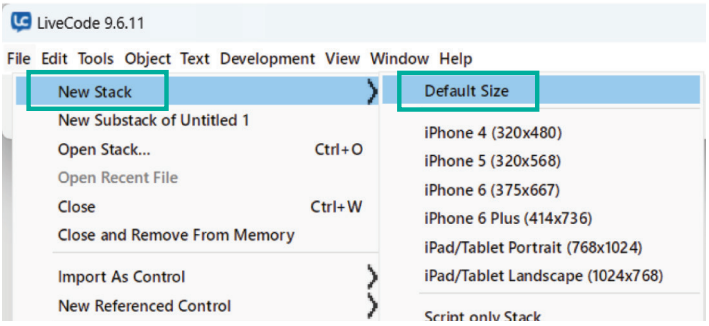


Fig. 2.14

Observación

Fíjate que LiveCode propone diferentes tamaños para la página o ventana principal, teniendo en cuenta diferentes tipos de dispositivos móviles (teléfonos y tabletas).

El nuevo proyecto contendrá un ***Main Stack*** que será algo así como la ***"página" principal.***

Nota: automáticamente se crea una pila con una única tarjeta con el tamaño definido.

2. Se diseñan objetos como botones de diversos tipos: campos, figuras geométricas; se importan recursos multimedia como sonidos, video,

etcétera, en diferentes tarjetas, con ayuda de la ***Paleta de herramientas (Tools Pallete).***

3. Se programa la interactividad con los objetos necesarios.
4. Se define la navegación entre las tarjetas.
5. Se determina para qué sistema operativo se trabaja.
6. Se compila y se obtiene el ejecutable de la aplicación.

Como recordarás, la paleta tienes dos saetas que permiten conmutar los modos de trabajo de ***LiveCode:*** el Modo de diseño o programación y el Modo de corrida o ejecución. De forma predefinida, al abrir la paleta esta se presenta en Modo de diseño o programación, lo cual nos permite diseñar objetos y programarlos. Una vez hecho esto, se podrá probar lo hecho para ver cómo funciona, entonces seleccionaremos la saeta de la izquierda ***(Modo de corrida o ejecución)*** y LiveCode pasará al modo usuario, lo cual nos permitirá probar lo hecho.

Nota: esta concepción permite ir probando el desarrollo de la aplicación en cada momento que se estime conveniente.

Los eventos, los objetos y sus propiedades en LiveCode

Ya con anterioridad se ha dicho que LiveCode es un lenguaje ***basado en objetos y dirigido por eventos.*** Esto quiere decir, entre otras muchas cosas, que el flujo del programa va a estar determinado por un sistema de ***acción y reacción.*** O sea, que para que se ejecute un programa en LiveCode, tiene que ocurrir algo que desencadene las acciones del programa en calidad de respuesta; eso que ocurre y desencadena acciones del programa se denomina ***evento.***

Las ***propiedades de los objetos*** son datos que conviven con los objetos y forman parte de estos. Se podría decir que las propiedades de los objetos determinan en ellos dos cosas:

- Su apariencia
- Su comportamiento

Ahora recordemos ¿cómo acceder o modificar las propiedades de un objeto? A las propiedades de un objeto se puede acceder de dos maneras, mediante:

- El inspector de propiedades
- El lenguaje de programación

El ***inspector de propiedades*** es un mecanismo del IDE o Interfaz de desarrollo que permite editar directamente las propiedades de un objeto. Para acceder al ***administrador de propiedades (Property Inspector)*** de un objeto basta con seleccionar el objeto, hacer clic derecho sobre él y seleccionar la opción ***administrador de propiedades (Property Inspector),*** como se muestra en la figura 2.15.

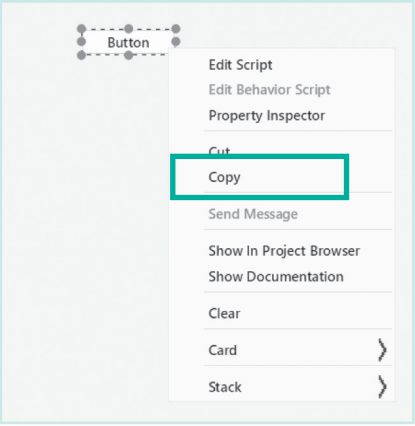


Fig. 2.15

Al acceder a la opción ***Property Inspector,*** aparecerá un cuadro de diálogo, este posee nueve secciones que invariantemente permiten acceder a diferentes propiedades de un objeto clasificadas por grupo, por ejemplo, el primer grupo son las propiedades más generales como: el identificador del objeto, su etiqueta (texto que se muestra), visibilidad, opacidad, estilo de borde, etcétera; el segundo grupo se refiere a lo que se denomina "Propiedades definidas por el usuario", aspecto que será tratado más adelante; el tercer grupo está dedicado a los colores de texto, de fondo, texturas, etcétera; el cuarto grupo trabaja los íconos que pueden acompañar a un botón; el quinto grupo se refiere al tamaño y la posición del objeto, y el sexto al tipo de letra y tamaño, así como a la alineación del texto. Los grupos restantes escapan del alcance de este libro.

La otra manera de modificar las propiedades de un objeto es mediante el uso del ***lenguaje de programación,*** para lo cual se usa la siguiente sintaxis:

Asignación de propiedades usando el lenguaje
Set the *properties* of *Object* to *Valor*
(Asígnale a la propiedad del Objeto el valor)

Nota: para referirnos a un objeto en LiveCode se usa la sintaxis ***Tipo + Nombre*** o ***Tipo + idNumber.***

Supongamos que hemos denominado ***Entrada*** a un botón y se quiere que este tenga color azul, entonces a nivel del lenguaje de programación se diría:
Set the *backgroundColor* of *button "Entrada"* to *blue*

Propiedades personalizadas o Propiedades definidas por el usuario

Como se ha dicho con anterioridad, el empleo de propiedades es una de las características de la programación orientada o basada en objetos. Igual se ha visto su importancia ya que una propiedad puede definir tanto la apariencia del objeto como su funcionalidad.

Recordemos que LiveCode posee cientos de propiedades en sus objetos, para resolver a través de ellas disímiles problemas; pero el usuario también puede definir nuevas propiedades y esto no tiene que resultarle complejo.

La sintaxis para definir una propiedad, que ya fue presentada con anterioridad, es la siguiente:

Set the *properties* of *Object* to *Valor*
(asignar la propiedad del objeto con el valor …)

Pues bien, ahora mediante un ejemplo se podrá crear la propiedad "MiEdad" y asignarle el valor 35 de la siguiente manera:

Set *MiEdad to field "Contenido"* to *35*

En tanto que la propiedad "MiEdad" no existe y la sintaxis aplicada es correcta, LiveCode automáticamente crea la propiedad "MiEdad" y le asigna el valor 35. A partir de ese momento el valor 35 forma parte del campo contenido y este valor persistirá, o sea, perdurará mientras no sea eliminado, no es volátil como las variables, que una vez que apagues el sistema informático su valor desaparece.

Las propiedades de usuario son también muy importantes, sobre todo porque:

- Permiten asociar datos a objetos con los que se relacionan (encapsulamiento).
- Brindan un nivel de acceso a los datos más rápido que con las variables.
- Protegen la información controlando la forma en que se lee o se escribe.

Para crear propiedades de usuario desde la interfaz de LiveCode, puedes proceder de la siguiente forma (figura 2.16):

1. Seleccionar el objeto al cual deseamos crearle la propiedad.
2. Dar clic derecho y acceder al inspector de propiedades.
3. Seleccionar el panel "Custom Properties" (segundo ícono).
4. Entrar un nombre para la nueva propiedad de usuario (***Key***).
5. Entrar un valor (***Value***).

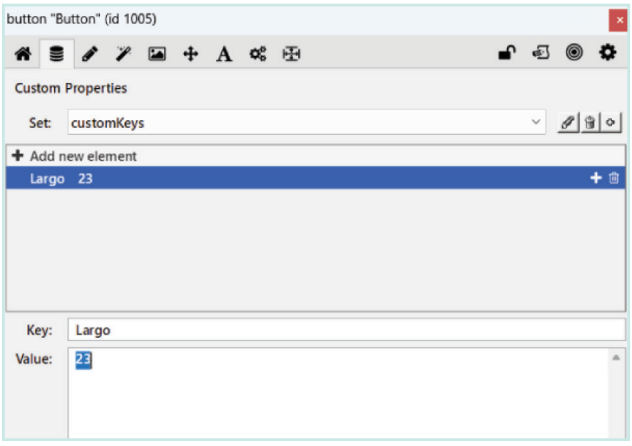


Fig. 2.16

Una vez asignada la propiedad de usuario a un objeto es posible recuperarla mediante la sentencia:

Put the Propiedad of Obj into Var

Si deseas saber qué propiedades de usuario posee un objeto, puedes seleccionar el objeto y usar la sentencia:

get customkeys of the selectedobject
put IT

Como estudiastes en año anteriores, a pesar de que LiveCode maneja más de 2 600 sentencias, bastarán dos de ellas (***Ask*** y ***Answer***), tres estructuras de control de flujo y los operadores estudiados, para implementar los algoritmos trabajados en este grado.

Al emplear una sentencia ***Ask,*** que permite la ***entrada*** por escritura de cualquier tipo de dato (numérico o alfanumérico), aparecerá un cuadro de diálogo como el que te muestra la figura 2.17. Esta instrucción será nuestro equivalente a ***Leer en pseudocódigo.***

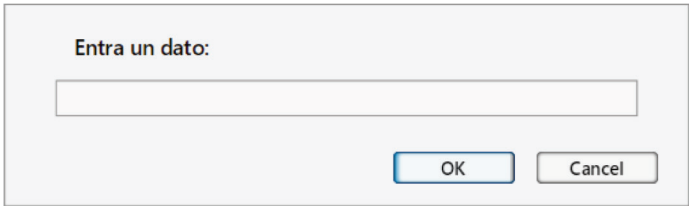


Fig. 2.17

Al emplear una sentencia ***Answer,*** se mostrará una caja de texto que permitirá la ***salida*** de cualquier tipo de dato (numérico o alfanumérico). Esta instrucción será nuestro equivalente a ***Escribir en pseudocódigo*** (figura 2.18).

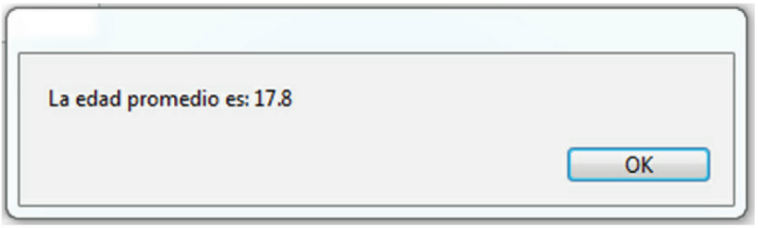


Fig. 2.18

2.4 Implementación de algoritmos

En este epígrafe pasaremos a la práctica, es decir, a una serie de ejemplos de cómo se implementan diferentes tipos de problemas algorítmicos en LiveCode.

Problemas secuenciales

1. Se necesita obtener el promedio de las notas en las asignaturas Matemática, Español e Historia de un educando de noveno grado.

```
/* Promedio de 3 notas */
On MouseUp
        Ask "Matemática"
        Put IT into Mat
        Ask "Español"
        Put IT into Esp
        Ask "Historia"
        Put IT into Hist
        Put (Mat + Esp + Hist) into Prom
        Answer "El promedio es "&Prom
End MouseUp
```

2. Elabora un programa que lea dos valores consecutivos de la lectura de un reloj contador y devuelva el importe a pagar por consumo energético.

Nota: se considera una familia ahorradora que consume menos de 100 Kw mensuales para la que el Kw/h cuesta 0,09 pesos.

```
/* Reloj-Contador */
On MouseUp
  Put "A pagar:"&&(field Lect2-field Lect1)*0.09 into field Importe
End MouseUp
```

Nota: observa que si el identificador de una variable se representa con más de una palabra, el uso de comillas entonces es obligatorio.

3. En una escuela de secundaria básica se efectúa un concurso de Ortografía. En la clave se plantea que por cada error de acentuación gráfica se quitarán 2 puntos, por errores de grafemas 4 puntos y por errores de uso de mayúsculas 5 puntos. Una vez calificados los exámenes, se conoce la cantidad de errores de cada tipo que fueron cometidos por cada educando. Elabore un algoritmo en pseudocódigo que permita calcular la cantidad de puntos que pierde cada educando.

```
/* Puntos ortográficos */
On MouseUp
        Ask "Entre la cantidad de faltas vinculadas con acentos."
        Put IT into Cant_acentos

        Ask "Entre la cantidad de faltas vinculadas con uso de grafemas."
        Put IT into Cant:Grafemas

        Ask "Entre la cantidad de faltas vinculadas con uso de mayúsculas."
        Put IT into Cant _May

        Put Cant_Acentos * 2 into Puntos_Acent
        Put Cant_Grafemass * 4 into Puntos_Grafemas
        Put Cant_May * 5 into Puntos_May

        Answer "Acentos:"&&Puntos_Acent&& "Grafemas:"&&
        Puntos_Grafemas&& "Mayúsculas:"&& Puntos_May
```

```
        Put Puntos_Acent + Puntos_Grafemas + Puntos_May into
        TotalPuntos

    Answer "Total:"&& TotalPuntos

End MouseUp
```

Nota: el empleo de doble ***ampersand*** (&&) garantiza que la concatenación se produzca dejando un espacio entre los operandos que se concatenan.

4. Como sabes, el área de un triángulo se calcula multiplicando la base del triángulo por su altura y dividiendo entre 2 el resultado de la multiplicación. Implementa un programa que calcule el área de un triángulo si se conoce su base y su altura.

```
/* Área de un triángulo */
On MouseUp
        Ask "Entre la base:"
        Put IT into VBase

        Ask "Entre la altura:"
        Put IT into VAltura

        Put VBase*Valtura/2 into VArea

        Answer "El Área del triángulo es:"&&Varea

           End MouseUp
```

5. Implementa un programa que calcule el área de un triángulo si se conoce la longitud de sus lados.

$$A = \sqrt{s(s - a)\ (s - b)\ (s - c)}$$

```
/* Área de un triángulo */
On MouseUp
        Ask "Entre el lado a:"
```

```
Put IT into A
Ask "Entre el lado b:"
Put IT into B
Ask "Entre el lado c:"
Put IT into C

Put (a+b+c)/2 into Semiper

Put sqrt(Semiper*(semiper-A)*(semiper-B)*(semiper-C)) into VArea

Answer "El Área del triángulo es:"&&Varea

End MouseUp
```

A continuación, se va a analizar un ejercicio donde se incorpora un nuevo elemento que son las ***listas.***

6. Se tiene un segmento AB ubicado en el primer cuadrante de un sistema de ejes cartesiano. Implementa un programa que calcule la distancia entre ambos puntos.
En este caso se va a programar de una manera diferente, pues vamos a basarnos en la propiedad de contenedores de información, en particular, campos para resolver el problema.

Nota: en LiveCode se denomina ***Contenedor (Container)*** a todo aquello que pueda almacenar datos (figura 2.19).

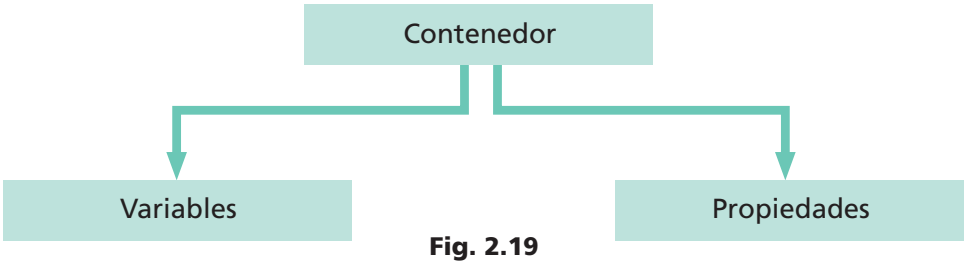


Fig. 2.19

Diseñemos un botón y tres campos con sus respectivas etiquetas: en dos de los campos Punto 1 y Punto 2, se escribirán las coordenadas ***X, Y,*** de los dos puntos cuya distancia se deberá calcular (figura 2.20).

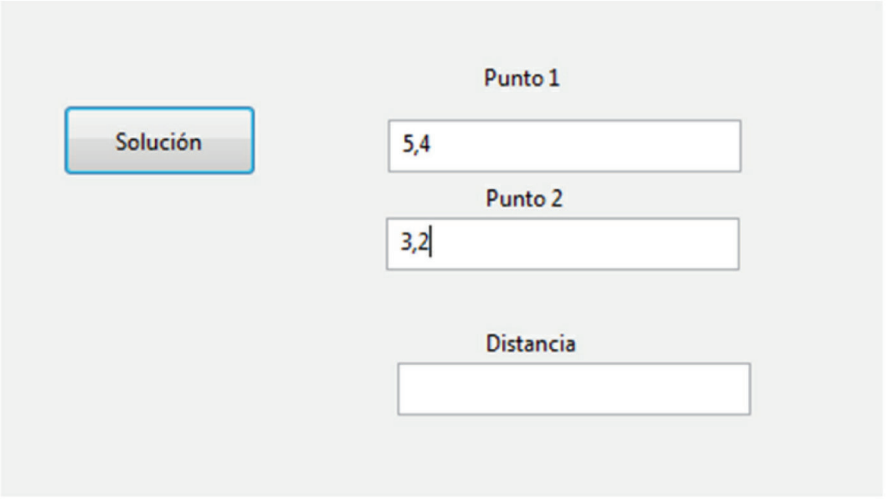


Fig. 2.20

Es importante observar que otro elemento de interés en este ejercicio son los datos estructurados, visto a partir del concepto de lista.

Definición

Una ***lista*** es una estructura de datos del tipo: ***a, b, c, d;*** o sea, un conjunto de datos separados por un delimitador. Implícitamente el delimitador es la coma (,) pero esto puede ser cambiado por conveniencia.

En nuestro ejercicio, las coordenadas de los puntos (5,4) y (3,2) pueden ser entendidas como dos listas de dos elementos cada una. Seguramente te preguntarás, ¿Cómo se puede acceder a los elementos de una lista?

Mediante el ***operador Item*** se puede acceder a los elementos de una lista, por ejemplo, si la lista se denomina L y sus valores son 5,4; entonces la sintaxis quedaría de la siguiente forma:
Put ítem 1 of L into xVar (Coloca un 5 dentro de la variable xVar)
Put ítem 2 of L into yVar (Coloca un 4 dentro de la variable yVar)

Ahora se verá la ***implementación del algoritmo de solución*** del ejercicio 6
/* Distancia */

```
on mouseUp
        Put item 1 of field "Punto 1" into x1
        Put item 2 of field "Punto 1" into y1

        Put item 1 of field "Punto 2" into x2
        Put item 2 of field "Punto 2" into y2

        Putsqrt((x2-x1)^2+ (y2-y1)^2) into Vdistancia

        Put VDistancia into field Distancia
        end mouseUp
```

Observaciones

1. En este ejercicio las sentencias ***Ask*** y ***Answer*** han sido reemplazadas en virtud del tratamiento de propiedades de objetos, en particular, ***campos (field).***
2. El operador ítem permitió extraer los componentes X, Y de los dos pares ordenados.
3. Como los nombres de los campos que contienen las coordenadas contienen espacios (Punto 1) (Punto 2), al referirnos al campo lo hacemos poniendo comillas delante y detrás, para connotar que se trata de una sola cadena.

Problemas alternativos

1. Dada la entrada de dos números, determina cuál es el mayor.

Nota: la interfaz del problema es un botón y tres campos: N1, N2 y Mayor.

```
/* Mayor de 2 */
On MouseUp
  Put field N1 into VN1
  Put field N2 into VN2
If VN1>VN2 then
  Put "El mayor es"&&VN1 into field Mayor
Else
  Put "El mayor es"&&VN2 into field Mayor
End if
End MouseUp
```

Ahora vamos a ampliar el problema de determinar el mayor de dos números a tres números.

2. Generaliza el algoritmo anterior para determinar el mayor entre tres números.

Nota: la interfaz del problema: un botón y cuatro campos: N1, N2, N3 y Mayor.

```
/* Mayor de 3 */
On MouseUp
  Put field N1 into VN1
  Put field N2 into VN2
  Put field N3 into VN3
 If VN1 >VN2 then
  Put Vn1 into MayorTemp
else
  Put Vn2 into MayorTemp
  End if

If MayorTemp > VN3 then
  Put MayorTemp into field Mayor
else
  Put Vn3 into field Mayor
End if
End MouseUp
```

A continuación, veamos otra variante de este mismo programa:

```
/* Mayor de 3 */
On MouseUp
  Put field N1 into VN1
  Put field N2 into VN2
  Put field N3 into VN3

If VN1 > VN2 and VN1 >VN3  then
  Put Vn1 into VMayor
```

```
End if
If VN2 > VN1 and VN2 >VN3  then
  Put Vn2 into VMayor
End if

If VN3 > VN1 and VN3 >VN2  then
  Put Vn3 into VMayor
End if

  Put VMayor into field Mayor
End MouseUp
```

Ahora se analizará otra variante de este mismo programa, pero en este caso se va a emplear un solo campo y una lista de tres números:

```
/* Mayor de 3 */
On MouseUp
  Put item 1 of field N into VN1
  Put item 2 of field N into VN2
  Put item 3 of field N into VN3

If VN1 >VN2 then
  Put Vn1 into MayorTemp
else
  Put Vn2 into MayorTemp
End if

If MayorTemp > VN3 then
  Put MayorTemp into field Mayor
else
  Put Vn3 into field Mayor
End if
End MouseUp
```

3. La historia de Cuba puede dividirse en etapas: Precolombina (antes de 1492), Etapa Colonial (1492-1858), Guerra de los Diez Años (1868-1878), Tregua Fecunda (1878-1895), Guerra Necesaria (1895-1898), Intervención

norteamericana (1898-1902), Neocolonia (1902-1958) y Revolución en el poder (1959-). Existe una base de datos que contiene "Efemérides", o sea, un compendio de hechos históricos con la fecha en que ocurrieron. La fecha se estructura de la siguiente forma: día ***D*** del mes ***M*** del año ***A.*** Elabora un algoritmo en pseudocódigo que al introducir una efeméride identifique la época histórica con la que se corresponde el hecho.

Para resolver este problema se va a introducir el equivalente de la instrucción ***Caso,*** que estudiamos en pseudocódigo. Esta estructura en LiveCode es:

```
  switch [Expresión]
case {Valor | Condición}
[instrucciones]
[default
instrucciones
end switch
```

A continuación, se explicará la sintaxis de esta complicada estructura:

- Las partes que se encierran entre corchetes ***[ ]*** son opcionales, o sea, pueden o no emplearse.
- La estructura se puede usar tanto con valores independientes como con condiciones
- Al finalizar cada bloque de instrucciones, se coloca una sentencia ***break*** para garantizar que no se mezclen los casos.

Veamos ahora la implementación del ejemplo citado:

```
/* Efemérides */
Ask "Entre año del acontecimiento: "
Put IT into VFecha
Switch
Case VFecha< 1492
answer "Precolombina"
break
Case VFecha>= 1492 and VFecha < 1868
answer "Colonial"
```

```
break
Case VFecha>= 1868 and VFecha < 1878
answer "Guerra 10 años"
break
Case VFecha>= 1878 and VFecha < 1879
answer "Tregua Fecunda"
break
Case VFecha>= 1895 and VFecha < 1898
answer "Guerra Necesaria"
break
Case VFecha>= 1898 and VFecha < 1902
answer "Ocupación norteamericana"
break
Case VFecha>= 1902 and VFecha < 1959
answer "Neocolonia"
break
Case VFecha>= 1959
answer "Revolución en el poder"
break
default
answer "Período no registrado"

endSwitch
End MouseUp
```

Observaciones

1. ***Last Word of It:*** devuelve la última palabra recogida por la instrucción ***Ask***, en el formato: acontecimiento - mes - año.
2. Es importante tener en cuenta el manejo de los signos >= y > para evitar que se solapen los intervalos.
3. ***Default*** cubre las variantes que queden fuera de los intervalos previstos, o sea, cuando no se cumple ninguna de las condiciones anteriores.

4. El programa /* Reloj-Contador */ que vimos en los ejemplos de problemas secuenciales es un tanto ficticio si se analiza su valor práctico, y el asunto estriba en que la tarifa de cobro de la electricidad no es

una función lineal, sino que es una tarifa progresiva; o sea, que en dependencia del consumo así será la tarifa que se emplee. La estructura ***switch*** que acabamos de estudiar es idónea para convertir el programa citado en un programa realmente útil. Veamos cómo sería su implementación ahora.

```
/* Reloj-Contador Real*/
On MouseUp
        Ask "Introduzca un consumo"
        Put IT into Consumo_1
        Ask "Introduzca el otro consumo"
        Put IT into Consumo_2
        Putabs(Consumo_1-Consumo_2) into Diferencia

        Switch
        Case Diferencia>= 0 and Diferencia<= 100
        answer "Importe"&&Diferencia*0.09
        break
        Case diferencia>=101 and diferencia<= 150
        answer "Importe"&&Diferencia*0.30
        break
        Case diferencia>=151 and diferencia<= 200
        answer "Importe"&&Diferencia*0.40
        break
        Case diferencia>=201 and diferencia<= 250
        answer "Importe"&&Diferencia*0.60
        break
        Case diferencia>=251 and diferencia<= 300
        answer "Importe"&&Diferencia*0.80
        break
        Case diferencia>=301 and diferencia<= 350
        answer "Importe"&&Diferencia*1.50
        break
        Case diferencia>=351 and diferencia<= 500
        answer "Importe"&&Diferencia*1.80
        break
        Case diferencia>=501 and diferencia<= 1000
```

```
        answer "Importe"&&Diferencia*2.00
        break
        Case diferencia>=1001 and diferencia<= 5000
        answer "Importe"&&Diferencia*3.00
        break
        Case diferencia>5000
        answer "Importe"&&Diferencia*5.00
        break
        endSwitch
End MouseUp
```

Observaciones

1. Nota que LiveCode es "No case sensitive", o sea, no sabe diferenciar implícitamente entre mayúsculas y minúsculas; pues para LiveCode es lo mismo ***Diferencia*** que ***diferencia*** que ***DifeRENCIA***.
2. El empleo del operador ***abs()*** garantiza que no sea importante el orden en que se introduzcan las lecturas, por lo que será igual escribir: ***última y anterior*** o ***anterior y última***.
3. Observa el manejo de los intervalos.

5. Elabora un programa que reciba un número del 1 al 10 en indo arábigo y devuelva su equivalente en romano.

```
/* Romano*/
On MouseUp
Ask "Entre el número en indo arábigo:"
Switch
        Case IT= 1
        Answer "I"
        Break

        Case IT= 2
        Answer "II"
        break

        Case IT= 3
```

```
        Answer "III"
        break

        Case IT= 4
        Answer "IV"
        break

        Case IT= 5
        Answer "V"
        break

        Case IT= 6
        Answer "VI"
        break

        Case IT= 7
        Answer "VII"
        break

        Case IT= 8
        Answer "VIII"
        break

        Case IT= 9
        Answer "IX"
        break

        Case IT= 10
        Answer "X"
    End switch
    End MouseUp
```

6. Implementa un algoritmo en LiveCode que transfiera la evaluación de un examen calificado en base a 100 en las calificaciones E, MB, B, R, M.

```
/*Escala*/
On MouseUp
```

```
    Ask "Entre la nota en base 100"
    Put IT into Nota
  Switch
   Case Nota>=90 and Nota<=100
     Answer "Excelente"
  break
   Case Nota>=80 and Nota<=90
     Answer "Muy Bien"
  break
  Case Nota>=70 and Nota<=80
     Answer "Bien"
  break
  Case Nota>=60 and Nota<=70
     Answer "Regular"
  break
  Default
    Answer "Insuficiente"
  End Switch
End MouseUp
```

7. Los once dígitos del carnet de identidad tienen un significado, como sabes, los seis primeros indican año, mes y día de nacimiento, y se dice que el último dígito tiene una significación especial: si es par, la persona es de sexo masculino y si es impar la persona es de sexo femenino. Elabora un algoritmo en pseudocódigo que al entrar el C. I. de una persona diga si es de sexo masculino o femenino.

```
/* Determina sexo */
/*CI*/
On MouseUp
      Ask "Entre el carnet de identidad"&&CI

      --- controla la cantidad de caracteres introducidos
      If the number of char of IT is not 11 then
      Answer "Cantidad incorrecta de cifras"
      BREAK
      End if
```

```
    Put last char of IT into Último_Dígito
    If Último_dígito mod 2 = 0 then
    answer "Sexo masculino"
    Else
    answer "Sexo femenino"
    end if
End MouseUp
```

Observaciones

1. En esta implementación aprovechamos para controlar que no exista equivocación en cuanto a la cantidad de cifras de un carnet de identidad, que como se sabe son once. Esto se logra mediante la sentencia: ***The number of chars*** (la cantidad de caracteres o letras de un contenedor).
2. Esta sentencia es aplicable a cualquier contenedor, o sea, a cualquier variable o cualquier propiedad.
3. También son aplicables las variantes:
   - ***The number of words*** (la cantidad de palabras de un contenedor o texto).
   - ***The number of lines*** (la cantidad de líneas de un contenedor o texto).
   - ***The number of ítems*** (la cantidad de ítems de un contenedor o lista).

8. Implementa un algoritmo en LiveCode que sea capaz de clasificar un triángulo a partir de sus lados.

Partiremos de la idea de que existen tres campos L1, L2 y L3, en los que se introducen los lados del triángulo y un cuarto campo para escribir la clasificación.

```
/* Tipo Triángulo */
On MouseUp

Put field L1 into VL1
Put field L2 into VL2
Put field L3 into VL3

Switch
```

```
    Case L1<>L2 and L1<>L3 and L2 <> L3
      Put "Escaleno" into field Clasifica
      Break

    Case L1=L2 and L1=L3
      Put "Equilátero" into field Clasifica
      Break

    Default
      Put "Isósceles" into field Clasifica
End switch

End MouseUp
```

Observaciones

1. En el segundo caso se pone de manifiesto lo que usualmente llamamos "carácter transitivo" o transitividad.
2. Fíjate que si se comprueba que L1=L2 y que L2=L3, entonces resultará cierto que L1=L3.
3. La sentencia ***Default*** nos libera de una condicional relativamente compleja buscando que dos de los lados sean iguales.

9. Ecuación de segundo grado:
Se partirá de la idea de que los tres coeficientes de la ecuación se escriben en un único campo mediante una lista de tres elementos: a, b, c.

Modelo matemático:

$$X_1 = - b + \frac{\sqrt{b^2 - 4ac}}{2a} \quad y \quad X_2 = - b - \frac{\sqrt{b^2 - 4ac}}{2a}$$

```
On MouseUp
  Put field Coeficientes into Lista
  Put item 1 of Lista into a
  Put item 2 of Lista into b
  Put item 3 of Lista into c
```

```
Put b^2 - 4*a*c into Delta
    Switch
    Case Delta > 0
      Answer "Dos raíces reales"
        Put (-b + sqrt(Delta)) / 2*a into X1
        Put (b + sqrt(Delta)) / 2*a into X2
        Answer "Raíz = "&X1 &&"Raíz = " & X2
    break
    Caso Delta = 0
      Answer "Una raíz doble"
    put - b / 2*a into X1
       Answer "Raíz ="&X1
    break
    default
     Answer "No hay raíces reales"
    End switch
End MouseUp
```

Problemas cíclicos

1. Implementa un algoritmo en LiveCode que defina los números pares existentes desde un ***número inicial*** hasta un ***número final.***

Nota: en esta ocasión, se ha complejizado un poco el ejercicio que se hizo en pseudocódigo, no obstante, como se observa en el ejemplo, se puede conocer la cantidad exacta de números que serán chequeados.

En este caso, se usará la estructura cíclica de LiveCode, ***Repeat with,*** cuya sintaxis es:

```
Repeat with var=inicio to/down to fin [step incremento]
        Acciones
End repeat
```

Observaciones

1. ***Var*** es lo que se llama variable de control de ciclo. Es una estructura de tipo contador y funciona para saber la cantidad de repeticiones que se van realizando.

2. ***Inicio*** y ***Fin*** son dos valores que marcan el inicio y el fin de las iteraciones.
3. ***Step incremento,*** como vez, se define de manera opcional por estar entre corchetes cuadrados ***[ ].*** Este valor permitirá indicar un valor de salto para el contador, o sea, este va a contar no de uno en uno como se hace implícitamente, sino de 2 en 2 o de 100 en 100, según sea el valor del incremento.
4. Como ves dice ***to/down to***, lo que significa que el conteo se puede hacer ascendente si se usa ***to*** o descendente si se usa ***down.***

```
/* Números pares entre 2 números */
On MouseUp

    Put 0 into C
    Ask "Entre el valor de inicio"
    Put IT into Vinicio
    Ask "Entre el valor de fin"
    Put IT into VFin

        Repeat with I = Vinicio to fin
            If I mod 2 =0 then
              Add 1 to C
            End if
        End repeat
    Answer "La cantidad de números pares es:"&&C
End MouseUp
```

2. Implementa un algoritmo en LiveCode que determine la nota promedio de un grupo de N educandos.

Nota: se partirá de la idea de que los nombres y las notas de los educandos se encuentran escritos en un campo denominado "Educandos", de la siguiente manera:

Juan Pérez 89
María Rodríguez Verdecia 98
Lucía Hernández Manzano 78
Katia Acosta 65

```
/*  Promedio */
On MouseUp
  Put 0 into VSuma
  Put the number of lines of field Educandos into VCant
  Repeat with i=1 to VCant
    Put last word of line I of field Educandos into Nota
    Add Nota to VSuma
  End repeat
  Put round(VSuma/VCant,2) into Promedio
  Answer Promedio
End MouseUp
```

Observaciones

1. La primera línea inicializa el acumulador ***VSuma.***
2. La línea de código "Put the number of lines of field Educandos into VCant" determina la cantidad de líneas escritas que posee el campo "Educandos", o sea, la cantidad de educandos registrados con sus notas.
3. ***Repeat with i=1 to VCant*** (abre la estructura repetitiva de tipo ***Repeat with***, en la que se sabe la cantidad de veces que se producirá el ciclo.
4. "Put last word of line I of field Educandos into Nota", coloca en la variable ***Nota*** la última palabra de la línea i, donde i va desde 1 hasta VCant, o sea, la primera vez la línea 1, las segunda vez la línea 2, etcétera.
5. ***Round*** es una sentencia u operador de tipo función que sirve para redondear el resultado de la división ***VSuma entre VCant,*** o sea, el promedio.

3. Implementa un algoritmo en LiveCode que lea lados de triángulos y calcule su perímetro tantas veces como sea necesario hasta que el perímetro calculado sea mayor o igual a 20 unidades.

Nota: como se observa en el ejemplo, no se conoce la cantidad exacta de veces que leeremos los lados del triángulo y no es posible saberlo; por tal motivo, no se puede usar la estructura de tipo ***Repeat With;*** entonces se tendrá que usar la estructura de tipo ***Repeat While.***

```
/* Perímetro */
On MouseUp
```

```
Ask "Pimer lado"
Put IT into LA
Ask "Segundo lado"
Put IT into LB
Ask d "Tercer lado"
Put IT into LC

Put La+Lb+Lc Into VPerimetro
answer VPerimetro
        Repeat while Vperimetro < 20
        Ask "Pimer lado"
        Put IT into LA
        Ask "Segundo lado"
        Put IT into LB
        Ask "Tercer lado"
        Put IT into LC
        Put La+Lb+Lc Into VPerimetro
        answer VPerimetro
        Endrepeat
EndMouseUp
```

4. Implementa un algoritmo en LiveCode que se use en la entrada de un cine y compute tres posibles causas que promuevan el interés de ver la película. Se introducirá ***D*** "si el interés es deseo de ver un Drama", ***P*** "si el interés es el país de origen de la película" y ***M*** "por la banda sonora de la película". Se desea calcular los porcientos en que se manifiestan cada uno de los intereses.

Nota: por las características del problema, está claro que no se sabe a priori la cantidad de personas que asistirán al cine, por tal motivo será necesario determinar un valor de entrada para detener el ciclo y mostrar los resultados.

```
/* Preferencias */
On MouseUp
Put 0 into CuentaDramas
```

```
Put 0 into CuentaPais
Put 0 into CuentaMusica
Repeat while Preferencia <> "Fin"
    Ask "Preferencia; D-Drama, P-País, M-Música, Fin- Finalizar"
    Put IT into Preferencia
    Switch
    Case Preferencia=D
    Add 1 to CuentaDramas
    break
    Case Preferencia=P
    Add 1 to CuentaPais
    break
    Case Preferencia=M
    Add 1 to CuentaMusica
    break
    Endswitch
EndRepeat
answer "Por Drama"&& CuentaDramas
answer "Por el país"&& CuentaPais
answer "Por la música"&& CuentaMusica
End mouseUp
```

Mensajes y funciones de LiveCode

Ahora se pasará a estudiar otros elementos importantes en LiveCode, que son los ***mensajes y funciones.***

Mensajes

Como se ha explicado con anterioridad, LiveCode es una herramienta de desarrollo basada en objetos y dirigida por eventos, en este acápite se va a profundizar en lo concerniente a los eventos, lo que abrirá enormes posibilidades prácticas de programación, en particular, en lo concerniente a la reusabilidad del código, lo cual a su vez redundará en una programación más eficiente.

Como se ha puesto de manifiesto en este libro, en ocasiones sentimos la sensación de que algo que estamos haciendo, ya lo hemos hecho antes y, por tanto pensamos: "si pudiera decir… haz esto o lo otro que ya hicimos en tal momento…", esto es ***reusabilidad.***

Para poder contar con esta posibilidad, se debe antes entender la filosofía de la programación dirigida por eventos y la jerarquía en que estos eventos se ponen de manifiesto.

Un evento es una ***acción*** ante la cual LiveCode responde mediante una ***reacción.*** Por eso se dice que sistemas como LiveCode tienen interfaces reactivas. Las acciones pueden provenir por dos vías:

- El usuario hace algo que provoca una reacción del sistema.
- El sistema hace algo que a su vez provoca una reacción del propio sistema.

En cualquiera de los casos las acciones producidas, ya sea por el usuario o por el sistema, se denominan ***mensajes.***

Definición

Un ***mensaje*** es una instrucción que se le envía a un objeto para notificarle que un evento ha ocurrido.

Esto significa que entre los eventos y los mensajes existe una relación biunívoca, o sea, para cada evento existe un mensaje y viceversa.

Ejemplo:
Si se hace "clic con el botón izquierdo del ratón" (evento) sobre un objeto, esto produce ***un mensaje*** a este objeto que se considera como una acción sobre el objeto.

Definición

Si el objeto que recibe el mensaje (evento) posee un código apropiado para dar respuesta al mensaje que recibe, este código se denomina ***manipulador de mensaje*** o ***manipulador del evento.***

Ejemplo:
Cuando se utiliza la siguiente línea de código ***On MouseUp, acciones End MouseUp,*** estamos utilizando un manipulador que en este caso es correspondiente al evento "hacer cilc con el botón izquierdo del ratón".

Si un determinado manipulador se coloca en un objeto, entonces el objeto estará listo para dar una respuesta o provocar una reacción ante el mensaje recibido.

Por ello, al solucionar los ejercicios propuestos en este epígrafe, se ha usado un código que se coloca en un botón que siempre contiene las sentencias siguientes:

- On MouseUp
- End MouseUp

Pues mediante estas dos sentencias (***Inicio-Fin***), hemos escrito las instrucciones (***acciones***) que dan respuesta al estímulo de haber hecho clic con el botón izquierdo del ratón (***implementación del algoritmo***).

Jerarquía de mensajes o Jerarquía de eventos

Por lo general, cada objeto de LiveCode es "hijo" o pertenece a otro objeto de jerarquía superior. Así, por ejemplo, un ***botón*** (***button***) es hijo o pertenece a una ***tarjeta*** (***card***), a su vez una tarjeta (***card***) es hija de una pila (***stack***), y esta, a su vez, pueden ser ***sub-stacks,*** hijos de otro ***stack*** que se denomina ***Main Stack.*** En fin, que en LiveCode, como en los seres humanos, existen parentescos.

Ahora medita un instante: ¿Qué sucede sobre un botón que se encuentra sobre una tarjeta y se le hace clic con el botón izquierdo del ratón?

Si el botón posee un manipulador apropiado para ese mensaje (***MouseUp***), entonces se ejecutarán las instrucciones que se encuentran en ese manipulador; pero si el botón no posee ese manipulador, el mensaje continuará en búsqueda del "padre" del botón, o sea, la tarjeta (***card***) intentando encontrar en el ***card*** o tarjeta el manipulador que no encontró el botón. Si lo encuentra, ejecutará las acciones que se encuentren dentro de este manipulador, pero si este manipulador tampoco existe, el mensaje buscará en el nivel superior, o sea, el "padre" de la tarjeta que es el ***stack*** o pila, como se muestra en la figura 2.21.

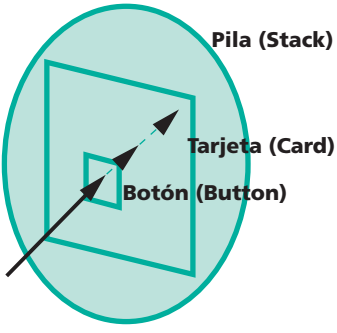


Fig. 2.21

Medita un instante de nuevo: ¿Cuál es el lugar idóneo para colocar un manipulador de mensajes: el botón, la tarjeta o la pila? ¿Existirá algún otro nivel superior?

Del análisis hecho se desprende una consecuencia muy importante, por lo que para dar respuesta a esta problemática, se deberá crear una situación hipotética. Imaginemos que tenemos una pila con 10 tarjetas o ***card*** sobre las cuales se definen campos (***field***) con textos, y se supone que existe un manipulador que responde al clic derecho del ratón y que indica automáticamente todos los errores ortográficos que posee el texto que se encuentra en el campo.

En el ejemplo hipotético que se muestra en cada tarjeta hay dos campos de texto. Si se parte de la idea de colocar el manipulador en cada campo, entonces se tendrían que colocar 20 manipuladores, uno para cada campo. Si se coloca un manipulador en cada tarjeta, este manipulador atendería los dos campos que se encuentran sobre esta, o sea, se reduciría de 20 manipuladores a 10; sin embargo, si se coloca el manipulador en el stack o pila de 10 manipuladores, se pasaría a un único manipulador, que atendería a todas las tarjetas y que a su vez atendería a todos los campos, o sea, se reduciría el código de 40 manipuladores a un manipulador.

En ***conclusión,*** el lugar idóneo para colocar un manipulador depende del nivel de generalidad en que se desea que se produzca la respuesta deseada. Mientras más alta la jerarquía más alta será la generalidad de la respuesta.

Nota: esta ley debe ser utilizada con mesura, ya que puede producir efectos indeseados.

Por ejemplo, en el caso anterior, si se coloca el manipulador que corrige la ortografía al hacer clic con el botón izquierdo a nivel de ***stack,*** y se hace clic sobre una imagen que se encuentre sobre una tarjeta, el manipulador "intentará corregir la ortografía de la imagen", lo cual resultará absurdo.

Recuerda que...

1. Lo referente a la jerarquía de mensajes desde LiveCode puede ser aún más profundo de lo que hemos explicado y, sobre esto, es posible leer en la documentación del lenguaje.
2. Si se quiere que varios manipuladores den respuesta a un mensaje, es posible que un manipulador le pase el mensaje a un manipulador de jerarquía superior y esta posibilidad podría ser empleada en situaciones sofisticadas. La sentencia que permite hacer esto es la sentencia ***Pass.***

Mensajes definidos por el programador

Como se ha visto con anterioridad, LiveCode está construido de manera tal que es capaz de interpretar un gran conjunto de mensajes que pueden ser provenientes de acciones del usuario o del propio sistema.

Ejemplos de algunos de estos mensajes son los que se proponen en la tabla 2.4.

Tabla 2.4

Mensaje	Acción
MouseUp	Hacer clic con el botón izquierdo del ratón.
MouseDoubleUp	Hacer doble clic con el botón izquierdo del ratón.
MouseEnter	El cursor del ratón intercepta el área de un objeto al entrar.
MouseLeave	El cursor del ratón intercepta el área de un objeto al salir.
OpenCard	Se presenta una tarjeta.
CloseCard	Se abandona una tarjeta.
Openstack	Se abre una pila.
CloseStack	Se cierra una pila.
DragEnter	Se intercepta un objeto en una operación de "Arrastrar y Soltar".
KeyDown	Se oprime una tecla.

Estos son solo algunos de las decenas de mensajes que están ***predefinidos*** en LiveCode (***build-in***); sin embargo, el programador puede construir sus propios mensajes. A continuación, verás la sintaxis de construcción de manipuladores personalizados o definidos por el programador.

La sintaxis de un manipulador personalizado es la siguiente:

On Nombre del manipulador Par1, Par2, …., ParN
Acciones
End Nombre del manipulador

Nota: los nombres o identificadores de los manipuladores no pueden llevar espacios.

Ahora imagina que quieres crear un mensaje que al enviarse provoque la animación por desplazamiento de un objeto cualquiera desde un punto ***P1,*** con coordenadas ***X1,Y1,*** hasta un punto ***P2*** con coordenadas ***X2,Y2.*** En el problema planteado vas a analizar qué es lo que puede variar.

Como se deduce, por lo pronto pueden variar cuatro cosas: el ***objeto*** que debe ser movido, las ***coordenadas de inicio*** del movimiento, las ***coordenadas del fin*** del movimiento y el ***tiempo que debe durar*** el movimiento.

Acorde con la sintaxis, este manipulador podría ser:

```
On Anima Obj,P1,P2,T
  Move Obj from p1 to p2 in T ticks
End Anima
```

Explicación:

1. ***Anima*** es el nombre del manipulador. En tanto este es un manipulador personalizado, su nombre es algo que define el programador; en este caso, se ha decidido que se llame ***Anima,*** pero pudiera haberse llamado ***LaAnimacion*** o ***MiAnimacion,*** etcétera.
2. Los elementos Obj,P1,P2 y T se denominan parámetros del mensaje y representan respectivamente:
   - Obj: el objeto que debe ser animado.
   - P1: las coordenadas x, y, del punto de inicio.
   - P2: las coordenadas x, y, del punto de fin.
   - T: el tiempo previsto que dure la animación expresado en ***ticks.***

Notas: 1 tick es igual a: 1/16 segundos.
Obj es igual a: Tipo + Nombre.

Acorde con lo planteado, el lugar idóneo para colocar un manipulador depende del nivel de generalidad en que se desea que se produzca la respuesta deseada. Si se quiere que este mensaje esté disponible desde cualquier tarjeta, lo correcto será colocarlo a nivel de ***stack*** o pila.

Con este objetivo, se procederá de la siguiente forma: ***Object – Stack Script,*** y redactaremos allí nuestro manipulador (figura 2.22), por ejemplo:

on Anima Obj,P1,P2,T
Move ***Obj from p1 to p2 in T ticks***
end ***Anima***

Luego, se podrá insertar la imagen que vamos a animar (figura 2.23) con la siguiente sintaxis:

File – Import As Control – Image File...

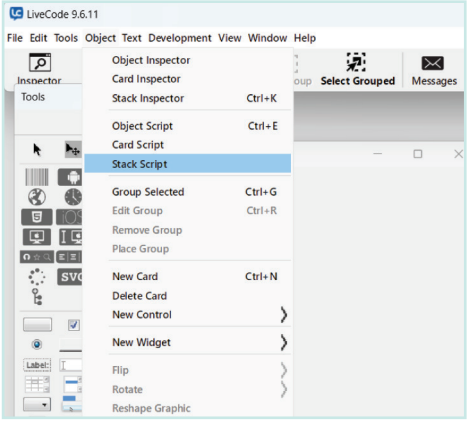


Fig. 2.22

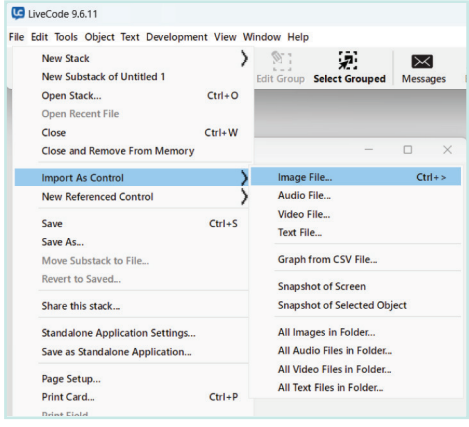


Fig. 2.23

Posteriormente, se buscará dentro del disco duro y encontraremos la imagen (figura 2.24), por ejemplo: "Koala.png".

La imagen se importará sobre la tarjeta (***card***) activa. Luego se va a diseñar un botón y en su código vamos a escribir el manipulador ***On MouseUp*** correspondiente, todo como se expone a continuación:

On ***mouseUp***
Put ***"image Koala.png" into Obj***
put ***"10,10" into P1***

```
put "200,200" into P2
Put 100 into T
  Anima Obj,P1,P2,T
end mouseUp
```

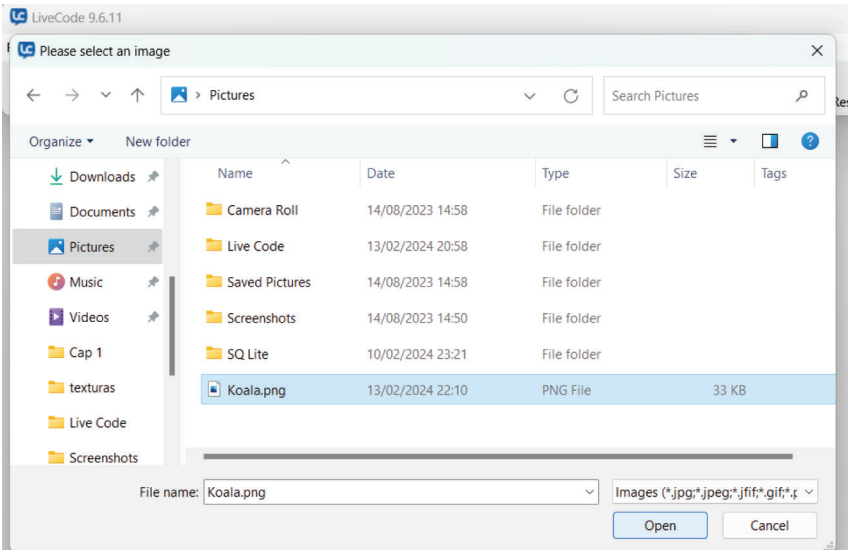


Fig. 2.24

Explicación:

1. Se asignan los valores respectivos a los parámetros previstos.
2. En tanto a nivel de ***stack*** existe un manipulador personalizado cuyo nombre es ***Anima,*** y a este se le pasan los parámetros Obj, P1, P2 y T; el mensaje ***Anima*** llegará hasta el nivel ***stack*** y allí se producirá la respuesta esperada, o sea, se animará la imagen.

Funciones

¿Recuerdas el concepto de función al estudiar Matemática? Se dice que una función en Matemática, es una relación entre dos conjuntos ***X*** y ***Y,*** de manera tal que a cada elemento del conjunto ***X*** o conjunto de partida, le corresponde uno y solo un elemento del conjunto ***Y***, o conjunto de llegada.

Ejemplos de funciones estudiadas son:

a) La función lineal $f(x)=ax+b$
b) La función cuadrática $f(x)=ax^2+bx+c$
c) La función valor absoluto $f(x)=|a|$

Desde el punto de vista informático, lo importante del concepto matemático es que como en toda función, para cada valor de ***X*** siempre existirá un valor ***Y***, o sea, que una función siempre ***expresa o devuelve un valor.***

Al igual que existen los mensajes predeterminados de LiveCode (MouseUp, KeyUp, MouseEnter, etcétera); también existen cientos de funciones predeterminadas (***build-in***) por LiveCode.

Ejemplos de estas funciones son las siguientes:
Abs(Valor) – Valor absoluto.
Average(Lista) – Promedio o media aritmética de una lista.
Sum(Lista) – Devuelve la suma de los valores de una lista.
Min(Lista) – Devuelve el menor valor de una lista.
Max(Lista) – Devuelve el mayor valor de una lista.

Como se puede analizar, en todos los casos hay algo en común y es el hecho de que todas estas expresiones ***devuelven un valor.***

Pero, ¿podemos construir nuestras propias funciones? La respuesta es positiva. De la misma manera en que pudimos definir nuestros propios mensajes en el subepígrafe anterior, resulta también posible definir nuestras propias funciones. La sintaxis para esto sería del modo siguiente:

Function Nombre de la función P1, P2, ..., Pn
Acciones
Return Valor
End Nombre de la función

Ejemplo 1

Tomemos por ejemplo el cálculo del área de un triángulo si se conoce el valor de sus tres lados a, b y c. El modelo matemático de este problema es:

$$A = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

La función Área de un triángulo "No existe en LiveCode", por tal motivo tiene sentido crearla. ¿Qué se necesita para el cálculo del área según este modelo matemático? Pues únicamente los valores de los tres lados del triángulo; luego, entonces, la función podría quedar definida de la siguiente forma:

```
Function Área_Triang a,b,c
 Put (a+b+c)/2 into Semiper
 Put sqrt(Semiper*(semiper - A)*(semiper - B)*(semiper - C)) into VÁrea
Return VÁrea
End Área_Triang
```

Si se coloca esta función en un nivel adecuado, por ejemplo, a nivel de ***stack,*** podrá hacerse uso de esta función desde cualquier control de la aplicación que se represente con este ***stack.*** Por ejemplo, desde un botón cualquiera, se podría hacer el llamado a la función de la siguiente forma:

```
On MouseUp
  Put Área(10,20,15) into Var
  Answer "El área es:"&&Var
End MouseUp
```

Otra variante podría ser la llamada a la función mediante el comando GET, como se expone a continuación:

```
On MouseUp
get Área(10,20,15)
  Answer "El área es:"&&it
End MouseUp
```

En este segundo caso, la función devuelve el valor sobre la variable del sistema denominada "IT".

Medita un instante:

Se desea saber si dadas las longitudes de tres segmentos, estos pueden forman o no un triángulo. Para resolver este problema, ¿qué sería pertinente crear, un mensaje personalizado o una función personalizada?

Este problema en matemática se denomina desigualdad triangular y su modelo matemático dice que si se cumple que: (a+b > c) o (a+c > b) o (b+c > a), entonces la unión de estos tres segmentos forma un triángulo. O sea, la suma de dos de sus lados tiene que ser mayor que el tercer lado.

Ejemplo 2

La solución de este problema devuelve un valor, el valor ***Verdadero, si se puede formar el triángulo*** o el valor ***Falso, si no es posible formarlo.*** Entonces lo idóneo en este caso es conformar una función.

Nota: las funciones que devuelven un valor lógico (verdadero o falso) se denominan funciones "buleanas" o simplemente ***Boolean.***

```
Function Hay_Triangulo a,b,c
  If (a+b > c)  or (a+c > b) or (b+c > a) then
    Return True
  Else
    Return false
  End if
End Hay_Triangulo
```

Ejemplo 3

Desde los ejemplos anteriores resueltos, se ha estado trabajando con la idea de encontrar el mayor entre dos números, luego entre tres, e inclusive se supo que LiveCode posee una función denominada ***Max,*** que encuentra el mayor valor de una lista de datos; pero, ¿te atreverías tú a construir esta función? Veamos:

```
Function Mayor L
Put the number of items of L into Cant
  Put item 1 of L into Mayor
  Put 1 into i
  Repeat for Cant-1 Times
    Add 1 to i
    If item I of L > Mayor then
       Put item I of L into Mayor
    End if
  End repeat
  Return Mayor
End Mayor
```

Explicación:

1. Como se ha visto, el parámetro que se le pasa a la función es una lista de N elementos (por ejemplo, L:= 2,5,8,9,13,5,7, etcétera).
2. La primera instrucción ***Put the number of items of L into Cant*** le asigna a la variable ***Cant*** la cantidad de elementos o ítems de la lista ***L.***
3. De manera artificial se parte de suponer que el primer elemento de la lista es el mayor (***Put ítem 1 of L into Mayor***).
4. Comienza el ciclo de comparación con el resto de los elementos de la lista (***Repeat - End repeat***) (¿Por qué Cant-1 veces?)
5. Se adiciona 1 a i para hacer el recorrido de ítem a ítem.
6. Se compara el ítem en curso con el valor del ***Mayor supuesto***, si el ítem en curso resulta mayor que el Mayor supuesto, entonces se reasigna a Mayor el ítem supuesto.
7. Finalmente, en la variable ***Mayor***, quedará el mayor de todos los números.

Alcance de las variables

Implícitamente las variables en LiveCode son locales a los manipuladores. Esto significa que su valor es válido solo dentro de un determinado manipulador (mensaje o función). Si se quiere dar a conocer el valor de una variable en más de un manipulador, entonces tendríamos que usar la instrucción ***Global.***

La instrucción ***Global*** se puede usar de dos maneras:

1. Declarar las variables globales al inicio del código, antes de declarar cualquier manipulador (Globalidad a nivel del ***script*** del objeto):

Ejemplo:

```
Global Var
On OpenCard
  Put 5 into Var
 End OpenCard

On MouseUp
  Answer Var
End mouseUp

On KeyUp Car
  Answer Var
End KeyUp
```

En el ejemplo previsto se declara como global la variable ***Var.*** Al presentarse la tarjeta (***card***) ***OpenCard***, se le asigna el valor 5 a esta variable y como se observa, en cualquiera de los restantes manipuladores es conocido el valor 5 de la variable ***Var.***

2. Declarar las variables globales dentro de los manipuladores que desean compartir el valor de la variable.

Ejemplo:

```
On OpenCard
  Global Var
  Put 5 into Var
 End OpenCard

On MouseUp
  Answer Var
End mouseUp

On KeyUp Car
  Global Var
Answer Var
End KeyUp
```

En este caso se declara como global la variable ***Var*** dentro del manipulador ***OpenCard*** y dentro del manipulador ***KeyUp,*** pero no se declara dentro del manipulador ***MouseUp,*** por tal motivo, el manipulador ***KeyUp*** conocerá el valor de ***Var,*** no así el manipulador ***MouseUp.***

Nota: esta segunda variante funciona además fuera del contexto del ***script*** de un objeto, o sea, entre ***card*** e inclusive entre ***stacks.***

Las variables compuestas

De manera un tanto intuitiva, en temas anteriores de este epígrafe se introdujo el concepto de ***lista*** en calidad de variable compuesta, y la lista resultaba una estructura de datos cómoda para ciertas circunstancias; sin embargo, si se realiza un mejor análisis de este elemento, podremos notar

que para otras circunstancias nos va a resultar un tanto incómoda.

Por ejemplo, imaginemos que queremos procesar datos relacionados con un grupo de educandos teniendo en cuenta sus nombres, apellidos, calificaciones en cada asignatura, sexo, edad, etcétera. Como verás expresar esta información mediante una lista resultará inoperante:

Put "Juan Rodríguez, 60,78,89,91,M,15,María Pérez,87,91,87,90,-F,17,Pedro Hernández, 65,79,90,91,M,17" into L

Si observas bien el ejemplo anterior, solo se han declarado tres educandos, pero si imaginamos que son los educandos de toda una escuela, o de todo el país, obviamente, este método para este tipo de problema resultaría inoperante. Una solución para este problema desde LiveCode y desde el resto de los lenguajes de programación, sería el concepto de ***arreglo.***

Definición

Arreglo: un arreglo es una estructura de datos en la cual en una misma variable se puede almacenar más de un valor.

Para diferenciar los valores en un arreglo se utiliza un elemento denominado ***índice del elemento*** o ***Key,*** el cual va a estar encerrado entre corchetes cuadrados [ ]. Este es un elemento que en LiveCode puede ser numérico o literal.

A continuación se muestran algunos ejemplos:

```
Put "Pepe" into Educando[1]
Put "Juan" into Educando[2]
Put "María" into Educando[3]
Put "Petra" into Educando[4]
```

Si luego se quiere saber qué dato se encuentra dentro de un elemento de arreglo, bastará con escribir:

```
Put Educando[3] y obtendremos María
Put Educando[1] y obtendremos Pepe
```

O también:
Get Educando***[1],*** lo cual colocaría en la variable IT a "Pepe"

Gráficamente el arreglo en memoria podría entenderse como se muestra en la tabla 2.5.

Tabla 2.5

1	Pepe	Educando[1]
2	Juan	Educando[2]
3	María	Educando[3]
4	Petra	Educando[4]

La función ***extents()*** devuelve el primer y último índice de un arreglo. En el caso anterior si se dice: Get ***extents(Educando),*** obtendríamos en IT la lista de 2 valores 1,4.

Algo particular de LiveCode es que los índices o llaves de los arreglos no solo pueden ser números, como en el caso estudiado, sino también pueden ser alfanuméricos, o sea, textos. Imaginemos un arreglo cuyo identificador es ***Notas,*** entonces en LiveCode es posible:

Put 100 into Nota["Matemática"]
Put 85 into Nota["Español"]
Put 94 into Nota["Física"]
Put 84 into Nota["Geografía"]

Y luego obtener la nota de una asignatura cualquiera diciendo:
Get Nota["Física"], lo cual coloca dentro de la variable IT el valor 94.

La función ***Keys*** devuelve los índices de un arreglo como este. Si se dice:
Put ***Keys(Nota) into Vindices,*** dentro de ***Vindices*** tendremos:

Matemática
Español
Física
Química

Arreglos de listas

Posterior al análisis de estos conceptos, podemos retomar nuestro problema inicial, o sea, el cómo representar los datos diversos de múltiples educandos de un aula, por ejemplo:

Put ***"Juan Rodríguez, 60,78,89,91,M,15,María Pérez,87,91,87,90, F,17,Pedro Hernández, 65,79,90,91,M,17"*** into L

La solución a este problema es el empleo de la estructura de datos conformada por un ***arreglo de listas,*** como se muestra a continuación:

Put ***"Juan Rodríguez, 60,78,89,91,M,15" into Educando[1]***
Put ***"María Pérez 87,91,87,90,F,17" into Educando[2]***
Put ***"Pedro Hernández, 65,79,90,91,M,17" into Educando[3]***
Etc.

Conocer el sexo de Juan sería entonces decir:

Put ítem 6 of Educando[1]

La edad de María sería:

Put ítem 7 of Educando[2]

Arreglos de N dimensiones

Hasta el momento se han visto arreglos de una sola dimensión, que también son denominados ***Vectores;*** sin embargo, pueden también definirse arreglos de dos, tres o más dimensiones. En particular, los arreglos de dos dimensiones se denominan ***Matrices.***

Con ayuda de un arreglo de tipo matriz, se podrán almacenar todos los datos que se encuentren en una tabla de doble entrada, o sea, de N filas y M columnas, un ejemplo se muestra en la tabla 2.6.

Tabla 2.6

Grupo 1	Grupo 2	Grupo 3	Grupo 4
María	Juan	Pedro	José
Lucía	Sofía	Ana	Juana
Antonio	Luis	Andrés	Katia

Para almacenar estos datos será suficiente tener en cuenta una matriz de dos dimensiones, o sea, de tres filas y cuatro columnas M[3,4].

Entonces, se diría:

Put "María" into M[1,1] --- Primera fila y primera columna
Put "Juan" into M[1,2] --- Primera fila y segunda columna
Put "Pedro" into M[1,3] --- Primera fila y tercera columna
Put "José" into M[1,4] --- Primera fila y cuarta columna

Put "Lucía" into M[2,1] --- Segunda fila y primera columna
Put "Sofía" into M[2,2] --- Segunda fila y segunda columna
Put "Ana" into M[2,3] --- Segunda fila y tercera columna

Si deseamos obtener a Pedro, diríamos:
Get M[1,3] lo cual colocaría a Pedro dentro de la variable del sistema IT

Nota: el concepto de matriz de dos dimensiones puede ser generalizado a N dimensiones.

Tratamiento de ficheros

Hemos visto hasta el momento diferentes maneras de almacenar y recuperar la información en dependencia de la complejidad del dato, desde la ***variable simple,*** también denominada ***escalar,*** hasta las ***variables compuestas*** constituidas por las ***listas y arreglos,*** y ***arreglos de listas,*** por lo que hasta el momento si se trata de manejar datos de pequeños volúmenes como los educandos de un grupo e inclusive los educandos de una escuela pequeña, tendríamos como representar esos datos; pero es obvio que si se tratara de

todos los educandos de un país, o todos los miembros de un sindicato, o un censo de población, las estructuras de datos previstas no serían operativas; por consiguiente, ante tales situaciones, se trabajaría con ***ficheros.***

Los ***ficheros*** pueden ser de diferentes tipos, por ejemplo: ficheros de texto, ficheros binarios, etcétera.

Ahora, abordaremos, ligeramente, el tratamiento de ficheros de texto, en particular ***la creación y lectura de ficheros.***

Creación y lectura de ficheros de texto

La ***creación de ficheros de texto*** involucra tres procesos:

- Abrir o definir el fichero para escritura.
- Escribir los datos en el fichero.
- Cerrar el fichero.

La sintaxis es:

```
Open file <Nombre de fichero> for Write
        Write <dato> to file <Nombre de fichero>
Close file <Nombre de fichero>
```

Ejemplo 1

Se supone que tenemos un objeto de tipo ***field*** denominado "Contenido", que contiene un texto y se desea que el texto se almacene en un fichero. La solución sería:

```
Open file "Mifichero.txt" for Write
  Write field "Contenido" to file "Mifichero.txt"
Close File "Mifichero.txt"
```

Notas:

1. Si no se declara un camino específico para el fichero Mifichero.txt, como es el caso del ejemplo, el fichero se crea en la misma carpeta en que se encuentra el programa que lo genera.

2. Si el programa no ha sido compilado, su carpeta implícita será: C:\Users\usuario\AppData\Local\RunRev\DocumentationCache\8_0_1_community\IDE

3. Si se quisiera crear el fichero en una carpeta específica, podría usarse la función del sistema ***specialFolderPath().***

Por ejemplo, para crear ***el fichero en el escritorio,*** la ejecución será del modo siguiente:

```
On MouseUp
    Put specialFolderPath("Desktop")&"/Mifichero.txt" into fichero
    Open file fichero for Write
    Write field Contenido to file fichero
    Close File fichero
End mouseUp
```

Nota: se puede consultar en "Ayuda" otros argumentos para la función ***specialFolderPath()*** diferentes de ***Desktop.***

La ***lectura de ficheros de texto*** involucra también tres procesos:

- Abrir o definir el fichero para lectura.
- Leer los datos en el fichero.
- Cerrar el fichero.

Ejemplo 2

Ahora veamos el proceso inverso, o sea, vamos a suponer que tenemos un fichero de texto en el escritorio y se desea visualizar su contenido en un campo de nuestra aplicación que se llama "Contenido". La solución sería la ***lectura de ficheros de textos.***

Por ejemplo, para leer un fichero de texto que se encuentra en el escritorio, la ejecución será del modo siguiente:

```
On MouseUp
    /* asignamos a la variable fichero el fichero que se va a leer que
       se encuentra en el escritorio */
    Put specialFolderPath("Desktop")&"/Mifichero.txt" into fichero

    /* abrimos el fichero en el modo lectura */
    Open file fichero for Read
```

```
    /* leemos el fichero hasta el final (End Of File) */
    read from file fichero until EOF

    /* asignamos el contenido leído en el campo denominado Con-
    tenido*/
    put IT into field Contenido
    Close File fichero
End MouseUp
```

Nota: es importante entender que al leer el fichero hasta el final (EOF) todo su contenido pasa a la variable IT del sistema, lo que permite procesar esta variable IT si fuera necesario, por ejemplo, línea a línea, palabra por palabra, o de carácter a carácter, por solo citar algunos ejemplos.

En ***conclusión,*** un nivel más sofisticado del manejo de datos se logra con el procesamiento a partir de la interacción de LiveCode con SGBD (sistemas de gestión de bases de datos), aspecto que será tratado a continuación.

2.5 Proceso de desarrollo desde la ingeniería del *software*

El Proceso de Desarrollo de Software (SDP, del término en inglés Sotftware Development Process) se puede dividir en siete pasos principales. Estos pasos deben llevarse a cabo al crearse cualquier proyecto de ***software*** y se resumen a continuación:

Análisis

Es una declaración sobre lo que su programa va a realizar. También incluirá una descripción de los pasos principales del problema.

Diseño

Esto implica diseñar tanto la ***interfaz*** de usuario como la estructura del ***código*** del programa, para lo cual se emplean generalmente diagramas de flujo o de pseudocódigos.

Implementación

La etapa de implementación implica ingresar el código del programa utilizando el editor de texto incorporado dentro del entorno de programación. Se usará LiveCode para crear nuestros programas.

Prueba

Las pruebas son una parte importante de cualquier proyecto. Las pruebas garantizan que su programa sea confiable y robusto en el sentido de que debería producir los resultados correctos y no fallar debido a entradas inesperadas.

Una buena práctica indica que se debe probar nuestro programa con tres conjuntos de datos de prueba. Estos son:

- Normal (datos aceptados dentro de un rango establecido).
- Extremo (datos aceptados sobre los límites).
- Excepcional (datos que no son aceptados).

.

Documentación

La documentación se produce generalmente en forma de una guía del usuario y una guía técnica. La guía del usuario le muestra cómo usar las funciones y características del ***software,*** mientras que la guía técnica le brinda información sobre cómo instalar el ***software*** y los requisitos mínimos del sistema.

Evaluación

Una evaluación suele ser una revisión que muestra que su programa es adecuado para su propósito, en otras palabras, hace exactamente aquello para lo que fue diseñado.

La evaluación también debe centrarse en la legibilidad de su código de programa; es decir, debe utilizar un código legible y entendible para todos, con el fin de que este pueda ser reutilizado por cualquier programador. Siempre debe asegurarse de que su programa sea legible haciendo lo siguiente:

- Uso de identificadores significativos para nombres de variables y matrices.
- Uso de la identificación correspondiente para significar los niveles de subordinación del código.

- Uso de comentarios internos (// Esta subrutina hará lo siguiente ...).
- Uso efectivo de espacios en blanco entre subrutinas para espaciar el programa.
- Sangría para mostrar el inicio y el final de cualquier estructura de control.

Mantenimiento

El mantenimiento se realiza al final del proyecto. No se le pedirá al educando que realice ningún tipo de mantenimiento en sus programas, pero este deberá conocer que estos aspectos forman parte del desarrollo del ***software.***

En los ejemplos de este libro, se hará énfasis en los elementos de ***Análisis, Diseño e Implementación,*** fundamentalmente. Ejemplos de proyectos son los siguientes:

1. Libro electrónico simple
2. Presentación multimedia
3. Animación
4. Juego
5. Base de datos Criterio

Proyecto 1. Creando un libro electrónico elemental

En la creación de un libro electrónico se va a proceder de la manera siguiente:

1. Creamos una pila (***stack***) y la redimensionamos.
2. Asignamos color al fondo de la aplicación modificando el color de relleno de la pila (***stack***).
3. Diseñamos un campo para el título del libro.
4. Diseñamos en la tarjeta un campo (se puede decidir o no que el campo sea deslizable (***scroleable***).
5. Ajustamos el tipo de fuente y color del texto que contendrá el campo, así como el interlineado y otras propiedades.
6. Diseñamos un panel de botones de navegación personalizados, entre estos botones están: ***Primero, Anterior, Siguiente, Último, Ir a..., Buscar.***
7. Compartimos el campo y el panel de navegación con todas las tarjetas.
8. Pegamos texto en el campo diseñado.
9. Creamos una nueva tarjeta y hacemos el paso 7 tantas veces como sea necesario en las diferentes tarjetas.
10. Eventualmente introducimos imágenes.

La interfaz debe quedar de la siguiente manera (figura 2.25).

Fig. 2.25

Proyecto 2. Gestionando una base de datos

En el capítulo anterior estudiaste lo relacionado con las bases de datos, ahora verás cómo se puede gestionar una base de datos desde LiveCode. Primero, es importante que recuerdes los aspectos que se desarrollan a continuación.

¿Qué es una base de datos? Es una colección de información organizada y estructurada a través de un conjunto de datos denominados registros. Por ejemplo, la información organizada de todos los educandos de una escuela.

¿Qué es un registro o artículo de la base de datos? Los ***registros,*** también denominados artículos de la base de datos, están compuestos por campos que constituyen la menor cantidad de información gestionable de manera directa de una base de datos. Ejemplo de un registro:

Nombre	Edad	Grado	Grupo	Matemática	Física	Historia	Español
Juan Pérez	17	11	A	89	94	83	100

Ejemplos de campos: ***Nombre, Edad, Grado, Grupo, Matemática, Física, Historia y Español***

¿Qué es una tabla? El conjunto de todos los registros de una base de datos se denomina ***tabla.*** Ejemplo de una tabla se muestra a continuación.

Nombre	Edad	Grado	Grupo	Matemática	Física	Historia	Español
Juan Pérez	17	11	A	89	94	83	100
María Rodríguez	18	11	B	91	89	98	89
Pedro Lucas	17	11	A	92	86	95	89
José Gómez	18	11	B	91	89	98	89

Nota: una base de datos puede tener una o varias tablas.

También es importante que sepas que LiveCode puede manejar bases de datos de diferentes tipologías, entre ellas las siguientes:

- Bases de datos nativas
- Bases de datos SQLite
- Bases de datos MySQL
- Bases de datos PostgreSQL
- Bases de datos ODBC
- Bases de datos Oracle database
- Bases de datos Valentina

Elegir un tipo específico de base de datos depende de una serie de factores, entre ellos los siguientes:

- Existencia previa o no de la base de datos.
- Tener en cuenta el SGBD que se ha instalado en tu dispositivo informático.
- Los conocimientos que tienes acerca de un SGBD específico.
- La cantidad de usuarios que simultáneamente pueden acceder a los datos.
- Si la base de datos será local o en línea.
- El volumen de datos que se va a manejar.

A continuación se propone un ejemplo de ***problema:***

En el proceso de Perfeccionamiento Educacional se sometió a consulta de los especialistas de los territorios y de la Academia de Ciencias de Cuba, los materiales docentes que se prepararon para la remodelación

del proceso de enseñanza-aprendizaje. Esta consulta generó una significativa información que se tuvo en cuenta para el perfeccionamiento de estos materiales. Ahora surge la siguiente interrogante, ¿sería factible elaborar una herramienta informática que optimice el proceso de toma de decisión a partir de los criterios existentes? A continuación se exponen algunas variantes de solución:

Variante 1. Empleo de una base de datos nativa de Livevode:
Es posible que hayas observado que la estructura de la base de datos que se describe puede ser modelada mediante una pila (***stack***) de varias tarjetas (***cards***), que tiene los mismos campos (***fields***) en cada tarjeta (***card***). Un campo de base de datos es como un campo en una tarjeta, y un registro es como una tarjeta en sí misma. Una pila configurada de esta manera puede actuar como una base de datos que contiene una tabla.

Análisis

a) Descomposición:
¿En cuántas partes se puede descomponer este problema? Una primera aproximación podría concebir el problema en tres partes:
- Creación de la base de datos
- Gestión de funcionalidades de la base de datos
- Gestión de reportes

b) Reconocimiento de patrones:
Existe una analogía perfecta entre los conceptos de tabla y ***stack*** o pila.

c) Abstracción:
¿Cuál sería la estructura de esta base de datos? La base de datos podría contener los siguientes campos esenciales:
- Grado
- Criterio
- Tipo de criterio
- Proceder
- Origen
- Observaciones

Una vez concebida la base de datos, será necesario obtener diferentes reportes como:

- Estadísticas por valoración de criterio
- Listado de criterios positivos
- Listado de criterios negativos
- Listado de criterios considerados "no procedentes"
- Listado de criterios por origen y tipo de criterio

Diseño (creación de la base de datos)

d) Diseño de la interfaz de usuario:

- Para crear la base de datos, se crea una pila y se define su resolución siguiendo los siguientes procedimientos:

  File – New Stack – Default Size

- Para redimensionar el tamaño de la pila (***stack***) se siguen estos pasos (figura 2.26):

  Object – Stack inspector – Sección (Position) – With – 800, Height – 600

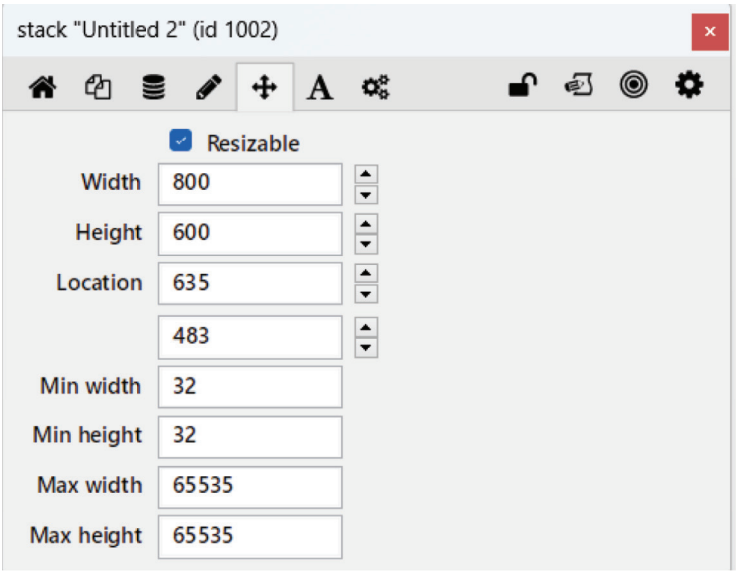


Fig. 2.26

- Cuando se crea una pila, se define en ella los cinco campos mencionados y los botones del panel de funcionalidades.

- Se diseñan los campos involucrados en la tarjeta que se presenta, que son los siguientes:
  - Grado
  - Criterio
  - Valoración - (Ortográfico/Contenido)
  - Ejecutar - (Procede/No procede)
  - Origen - (Lista de provincias)
  - Obs – Observaciones

- Se definen los botones del panel de funcionalidades:
  - Nuevo
  - Borrar
  - Guardar
  - Primero
  - Anterior
  - Siguiente
  - Último

- Se diseña una etiqueta para saber el número de artículo que está activo.

La interfaz debe quedar como se muestra en la figura 2.27.

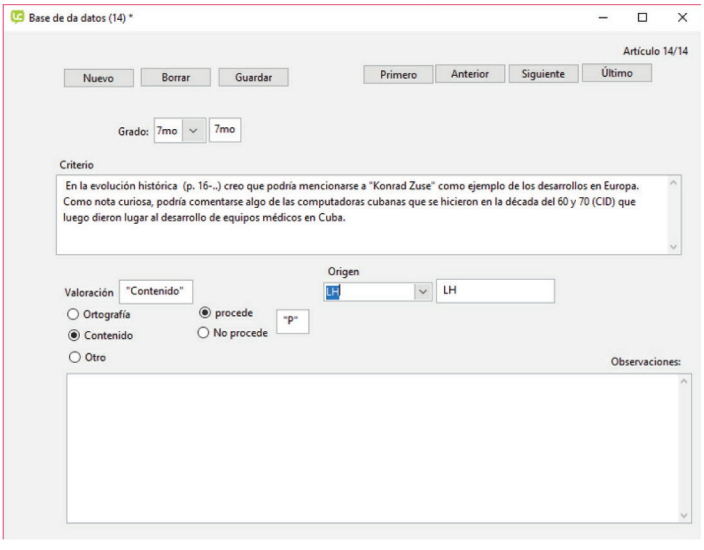


Fig. 2.27

Algunos aspectos que se deben tener en cuenta son los siguientes:

- Además de los objetos de tipo ***field*** (***campos contenedores de datos***), se usarán campos de tipo ***Label*** (***etiquetas***), para poder identificar los contenidos de los campos (letreros que acompañan a los campos).
- Los campos "Criterio" y "Observaciones" deben llevar ***scrolling*** vertical, ya que no se conoce ***a priori*** la cantidad de contenido que tendrán.
- Algunos campos como "Grado", "Valoración", "Ejecutar" y "Origen" poseen valores predecibles, por tal motivo resulta ergonómico (cómodo) y consistente que se definan mediante grupos de ***RadioButtons*** (***botones de radio***) y ***ComboBoxes*** (***listas desplegables***).
- Para que los ***RadioButtons*** sean mutuamente excluyentes es necesario que se agrupen.
- Los valores de las listas de los ***ComboBoxes*** se definen como propiedad de estos objetos.
- El panel de funcionalidades se define mediante siete botones estándares.

Aspecto fundamental

Para que la interfaz sea replicada en cada tarjeta (***card***) que se cree, una vez creada, en la primera tarjeta:

- Se seleccionan todos los elementos de la interfaz (campos, etiquetas y botones) y se agrupan ().
- Se le da un nombre al grupo creado.
- Se crea una segunda tarjeta que aparecerá vacía.
- Y se pega de manera especial el grupo creado (***Object – Place Group***).

De esta manera, toda vez que se cree una nueva tarjeta, se indicarán los objetos del grupo declarado, lo cual permitirá modelar la base de datos que deseamos.

Implementación

e) Implementación:

- Gestión de funcionalidades de la base de datos. Código de los botones de funcionalidades (tabla 2.6).

Tabla 2.6

Objeto	Función	Código	Observación
Botón ***Nuevo***	Crea un artículo vacío.	***onmouseUp*** ***gotolastcard*** ***NEWCard*** ***endmouseUp***	Se usará ***gotolastcard*** para forzar la acción de que una vez creada una nueva tarjeta, esta se cree después de la última.
Botón ***Borrar***	Elimina un artículo de la BD.	***onmouseUp*** ***answer "¿Estás seguro que deseas eliminar este artículo?" with "Sí" or "No"*** ***if it is Síthen*** ***delete this card*** ***Actualiza*** ***endif*** ***endmouseUp***	La operación es drástica, por tal motivo exige una confirmación. ¿Por qué es necesario invocar el mensaje ***Actualiza?*** ¿Cómo evitar que se produzca un error si al aplicar esta función solo existe una tarjeta?
Botón ***Guardar***	Guarda todos los artículos de la base de datos.	***onmouseUp*** ***save this stack*** ***answer "Base de datos guardada"*** ***endmouseUp***	Se envía un mensaje de retroalimentación para confirmar la operación.
Botón ***Primero***	Navega al primer artículo.	***onmouseUp*** ***go to first card*** ***endmouseUp***	
Botón ***Anterior***	Navega al artículo anterior.	***onmouseUp*** ***go to previous card*** ***endmouseUp***	
Botón ***Siguiente***	Navega al artículo siguiente.	***onmouseUp*** ***go to next card*** ***endmouseUp***	
Botón ***Último***	Navega al último artículo.	***onmouseUp*** ***go to last card*** ***endmouseUp***	

Combo Box***Grado***	Selecciona el grado a partir de una lista.	***onmenuPickpItem-Name putpItemName into field Grado endmenuPick***	El parámetro ***pItemName*** es la opción seleccionada de la lista.
Combo Box***Origen***	Selecciona el origen del comentario a partir de una lista.	***onmenuPickpItem-Name putpItemName into field Origen endmenuPick***	El parámetro ***pItemName*** es la opción seleccionada de la lista.
Grupo ***Tipo_Error***	Selecciona el tipo de error a partir de tres opciones.	***onMouseUp "put last word of name of target into field" Valoracion " endMouseUp***	Se usa ***lastword*** para lograr exclusivamente el nombre del objeto interactuado (***target***).
Grupo ***Hacer***	Selecciona el proceder con el comentario a partir de dos opciones.	***onMouseUp put last word of name of target into field "Ejecutar" endMouseUp***	Se usa ***lastword*** para lograr exclusivamente el nombre del objeto interactuado (***target***).
Stack ***Base_de_ datos***	Contiene procedimientos o manipuladores de mensajes globales.	***onOpenCard Actualiza endOpenCard onactualiza put the number of cards of this syack into CantRecords put "Artículo"&&the number of this card&"/"&Cantrecords into field Localizador end actualiza***	Se crea la acción ***actualiza*** como procedimiento para poderlo reutilizar en otros contextos.
Field ***Localizador***	Muestra la localización del artículo en curso.		Es el mensaje ***Actualiza***, el que modifica su propiedad.

Pruebas

f) Pruebas:

- Situaciones extremas.
- Eliminar quedando 1.
- Navegar al siguiente estando en la última.
- Navegar al anterior estando en la primera.

- Gestión de reportes

Para la implementación de los reportes, se puede emplear un segundo ***stack,*** que definitivamente se convertirá en el módulo principal de nuestra aplicación.

Variante 2

Análisis

a) Descomposición:

Esta tarea se puede dividir en dos momentos:

- Menú de selección de reportes y acceso a la base de datos.
- Salida de los reportes.

Diseño

b) Reconocimiento de patrones:

- Muchos reportes se van a diferenciar simplemente por el dato que se recupera.

c) Abstracción:

- Se pueden concebir dos tarjetas, una para el menú de opciones y otra para la salida de los reportes.

Diseño de la interfaz de usuario:

- Para crear la interfaz de reportes y acceso a la base de datos, se crea una pila y se define su resolución siguiendo los siguientes procedimientos:
  File – New Stack – Default Size

- Se redimensiona el tamaño de la pila (***stack***) siguiendo los siguientes pasos (figura 2.28):
  Object – Stack inspector – Sección (Position) – With – 800, Height – 600

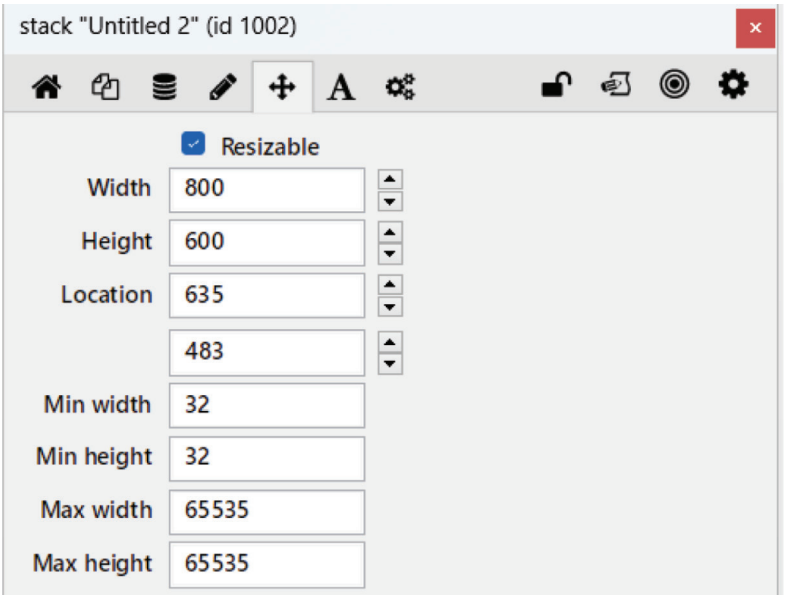


Fig. 2.28

- Se definen en la pila (***stack***) dos ***cards*** (***tarjetas***), una para el menú que vamos a denominar "Menú", y otra para los reportes que de manera similar, se van a denominar "Reportes".

- Se diseñan los objetos involucrados en la tarjeta ***Menú:***
  - Un ***field*** de tipo Etiqueta para poner título a la tarjeta ***Menú.***
  - Una imagen para ilustrar la tarjeta ***Menú.***
  - Un botón para Acceder a la base de datos.
  - Un botón para Estadísticas.
  - Un botón para Listados por tipo de criterio (Ortográfico/Contenido).
  - Un botón para Listados por resultados (Procede/No procede).
  - Un botón para Listados por origen (Lista de provincias).
  - Un botón para Criterios con observación.

- Se diseñan los objetos involucrados en la tarjeta ***Reporte:***
  - Botón regresar
  - Campo (***field***) Listado_Primario
  - Campo (***field***) Listado_Secundario

- Se diseñan etiquetas para los campos Primario y Secundario.

La interfaz debe quedar como se muestra en las figuras 2.29 y 2.30.

Fig. 2.29

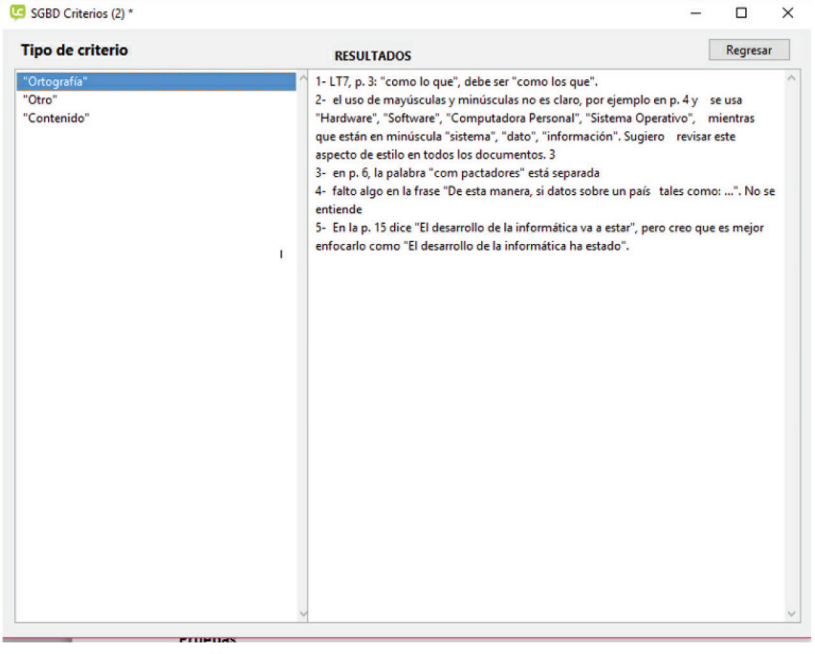


Fig. 2.30

Implementación

Algoritmos

- Gestión de funcionalidades del Menú principal. Código de los botones de la tarjeta Menú principal (tabla 2.7).

Tabla 2.7

Objeto	Función	Código	Observación
Botón ***Estadísticas***	Muestra las estadísticas de la base de datos.	***GlobalTipoReporte*** ***onmouseUp*** ***Put "Estadísticas" into field Titulo of Card Reporte*** ***Put "estad" into TipoReporte*** ***go to card Reporte*** ***endmouseUp***	
Botón ***Listado por tipo de criterio***	Clasifica los criterios y muestra la lista de criterios por cada categoría.	***GlobalTipoReporte*** ***onmouseUp*** ***Put "Tipo de criterio" into field Titulo of Card Reporte*** ***Put "Tipo_criterio" into TipoReporte*** ***go to card Reporte*** ***endmouseUp***	
Botón ***Listado por resultados***	Muestra el listado de los criterios según las categorías "Procede" y "No Procede".	***GlobalTipoReporte*** ***onmouseUp*** ***Put "Ejecución" into field Titulo of Card Reporte*** ***Put "Tipo_Proceder" into TipoReporte*** ***go to card Reporte*** ***endmouseUp***	

Objeto	Función	Código	Observación
Botón ***Listado por origen***	Muestra el listado de los criterios según el origen de procedencia	***GlobalTipoReporte onmouseUp Put "Origen del criterio" into field Titulo of Card Reporte Put "Tipo_origen" into TipoReporte go to card Reporte endmouseUp***	
Botón ***Criterios con observación***	Muestra el listado de los criterios que poseen observaciones	***GlobalTipoReporte onmouseUp Put "Observaciones" into field Titulo of Card Reporte Put "Obs" into TipoReporte go to card Reporte endmouseUp***	

- Código de la tarjeta Reportes
  /* El código de la tarjeta contiene todos los procedimientos y funciones necesarios para producir todos los reportes del proyecto */:

```
Global TipoReporte, CantRecords,ElCampo
onOpenCard
IniciaEscenario
Switch
Case TipoReporte = "Estad"
MuestraCantidad
        Grados
        Valoraciones
        Procede
        Origen
break
```

```
Case TipoReporte = "Tipo_criterio"
put "Valoracion" intoElCampo
        Lista ElCampo
break
Case TipoReporte = "Tipo_Proceder"
put "Ejecutar" intoElCampo
        Lista ElCampo
break
Case TipoReporte = "Tipo_origen"
put "Origen" intoElCampo
        Lista ElCampo
break
Case TipoReporte = "Obs"
ListaObservacion
endSwitch

endOpenCard

OniniciaEscenario
Put "" into field "Listado_Primario"
Put "" into field "Listado_Secundario"
endIniciaEscenario

OnMuestraCantidad
Put "Cantidad de criterios:"&&Cantidad() &Returnafterfield
"Listado_Primario"
endMuestraCantidad

functionCantidad
Put the number of cards of stack "Base_de_datos" into CantRecords
returnCantRecords
endCantidad

OnGrados
Put 0,0,0,0,0,0,0,0,0,0,0,0 into LGrados
put 0 into i
RepeatCantRecords Times
```

```
add 1 to i
put char 1 of field Grado of card i of stack "Base_de_datos" into
VGrado
add 1 to item VGrado of LGrados
endRepeat
put 0 into i
PutReturn after field "Listado_Primario"
Repeat 12 Times
add 1 to i
Put "Del grado"&&i&" = "& item i of LGrados&Return after field
"Listado_Primario"
endRepeat
endGrados

OnValoraciones
--- Contenido, Ortografía, Otros
put 0 into i
put 0 into Cant_Contenido
put 0 into Cant_Orto
put 0 into Cant_Otros

RepeatCantRecords Times
add 1 to i
Put field "Valoracion" of card i of stack "Base_de_datos" into La-
Valoracion

Switch
CaseLaValoracion Contains "Contenido"
add 1 to Cant_Contenido
break
CaseLaValoracion Contains "Ortografía"
add 1 to Cant_Orto
break
default
add 1 to Cant_Otros
endSwitch
endRepeat
```

```
Put Return after field "Listado_Primario"
Put "De contenido"&&Cant_Contenido&Return after field "Listado_Primario"
Put "De ortograflia"&&Cant_Orto&Return after field "Listado_Primario"
Put "Otros"&&Cant_Otros&Return after field "Listado_Primario"
end Valoraciones

On Procede
--- Contenido, Ortografía, Otros
put 0 into i
put 0 into Cant_Procede
put 0 into Cant_No_Procede

Repeat CantRecords Times
add 1 to i
Put field "Ejecutar" of card i of stack "Base_de_datos" into VProcede

if VProcede Contains "NP" then
add 1 to Cant_No_Procede
else
add 1 to Cant_Procede
endif
endRepeat

Put Return after field "Listado_Primario"

Put "Proceden:"&&Cant_Procede&Return after field "Listado_Primario"
Put "No proceden:"&&Cant_No_Procede&Return after field "Listado_Primario"
end Procede

On Origen
Put "PR,AR,LH,MAY,MT,VC,CA,SS,CM,LT,HO,GR,SC,GT,IJ,Academia" into LProv
Put 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 into L_Cant_Prov
```

```
Put the number of items of LProv into CantProv
put 0 into i
RepeatCantRecords Times
add 1 to i
put field Origen of card i of stack "Base_de_datos" into VOrigen

add 1 to item PosOrigen(LProv,VOrigen) of L_Cant_Prov
endRepeat
put 0 into i
PutReturn after field "Listado_Primario"

RepeatCantProv Times
add 1 to i
Put "Del origen "&&item i of Lprov&" = "& item i of L_Cant_Prov&Re-
turn after field "Listado_Primario"
endRepeat
end Origen

functionPosOrigenLProv,VOrigen

put the number of items of Lprov into Cantidad
put 0 into i
repeatCantidad Times
add 1 to i
ifVOrigen = item i of Lprovthen
return i
endif
endrepeat
-----return 0
endPosOrigen

---- ==============================
OnListaElCampo
Put "" into  field "Listado_Primario"
put 0 into i
put 0 into Cant_Procede
```

```
put 0 into Cant_No_Procede
RepeatCantRecords Times
add 1 to i
if field ElCampo of card i of stack "Base_de_datos"  is not in field
"Listado_Primario" then
put field ElCampo of card i of stack "Base_de_datos" &Return after
field "Listado_Primario"
endif
endrepeat
endLista

onListaObservacion
Put "" into  field "Listado_Primario"
Put "" into  field "Listado_Secundario"
put 0 into i
put 0 into C
RepeatCantRecords Times
add 1 to i
if field "Obs" of card i of stack "Base_de_datos"  is not "" then
add 1 to C
put C&"-"&&field "Criterio" of card i of stack "Base_de_datos" &Re-
turn&"OBSERVACIÓN"&Return& field Obs of card i of stack "Base_
de_datos"&Return after field "Listado_Secundario"
endif
endrepeat
endListaObservacion
```

Pruebas

Probar todas las funcionalidades.

Bases de datos no nativas

Con la biblioteca de la base de datos LiveCode, es posible comunicarse con las bases de datos externas más comunes en la actualidad, por ejemplo: MySQL, SQLitePostgreSQL, entre otras. También pueden obtenerse datos de bases de datos de un solo usuario o de multiusuarios, actualizar datos en ellas, obtener información sobre la estructura de la base de datos y mostrar datos de la base de datos en su aplicación.

Para comprender completamente este tema, es importante conocer cómo escribir códigos cortos y los conceptos básicos de las bases de datos. así como que LiveCode soporta directamente las siguientes implementaciones de base de datos, por ejemplo: ***MySQL; SQLite; PostgreSQL.*** LiveCode también admite la conexión a una base de datos a través de ODBC. Puede usar ODBC para acceder a Access, FileMaker, MS SQL Server y muchas otras implementaciones de bases de datos.

Los comandos y las funciones de la base de datos de LiveCode usan la misma sintaxis, independientemente del tipo de base de datos a la que se conecte; no es necesario aprender un lenguaje de base de datos separado para cada tipo. En cambio, cuando se abre por primera vez una base de datos con la función ***revOpenDatabase,*** se debe especificar su tipo como uno de los parámetros para que LiveCode sepa con qué tipo de base de datos está tratando. La biblioteca de la base de datos maneja los detalles de cada tipo de manera trasparente para el programador.

Software para acceso a bases de datos

Para proporcionar conectividad a las bases de datos, LiveCode funciona con controladores de base de datos: ***software*** que traduce las solicitudes de aplicaciones al protocolo requerido por una base de datos específica.

Los controladores de base de datos para ciertos tipos de bases de datos se incluyen en la distribución de LiveCode (la lista de tipos de base de datos incluidos depende de la plataforma).

Si instalaste LiveCode, tienes todo el ***software*** necesario para usar los tipos de base de datos incluidos. Para otros tipos de bases de datos, deberás obtener los controladores de base de datos adecuados antes de poder trabajar con esas bases de datos.

Entre los controladores de base de datos para trabajar con LiveCode están:

MySQL:

Los controladores de base de datos MySQL se incluyen como parte de la instalación de LiveCode en sistemas Linux, Mac OS X y Windows.

PostgreSQL:

Se incluye un controlador de base de datos PostgreSQL como parte de la instalación de LiveCode en sistemas Linux, Mac OS X y Windows.

SQLite:

Los controladores para acceder a esta base de datos se incluyen con LiveCode. No es necesaria ninguna instalación adicional.

Gestores ODBC y controladores de base de datos

Para utilizar una base de datos a través de ODBC, el usuario debe instalar el ***software***_ODBC necesario para su plataforma (algunos sistemas operativos incluyen una instalación de este tipo). El ***software*** ODBC incluye uno o más controladores de base de datos, además de una utilidad de administrador ODBC.

Los sistemas Windows incluyen el paquete MDAC (Microsoft Data Access Components) como parte de la instalación estándar del sistema. Para configurar ODBC en sistemas Windows, se usa el panel de control "Orígenes de datos ODBC".

Compilación de proyectos

Como se planteó con anterioridad, LiveCode es una herramienta libre, multiplataforma y multidispositivo, esto es consecuente con la consigna "***Write once, run any where***", o sea, se escribe el código una vez y se compila para cualquier sistema operativo.

La obtención de un producto ejecutable para cualquier plataforma (***stand alone***) consta de dos pasos:

1. Configurar la salida (en el menú ***File – Stand alone Application Setting***) como se muestra en la figura 2.31.
2. Generar el ejecutable (***Save as Stand Alone Application***).

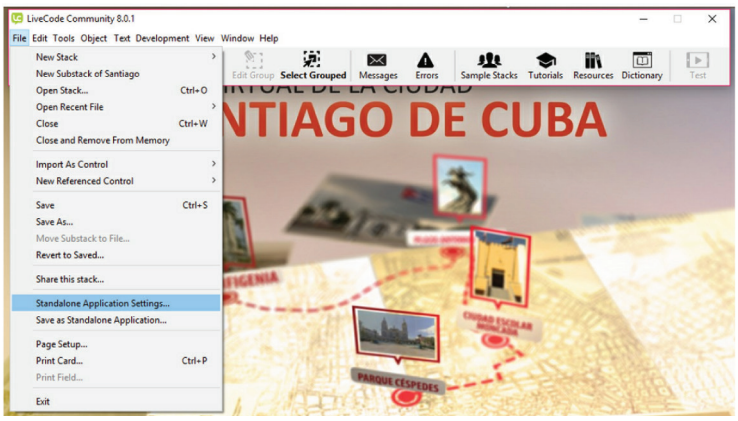


Fig. 2.31

Al entrar en la opción de ***Configuración de la salida,*** se presenta el cuadro de diálogo de la figura 2. 32.

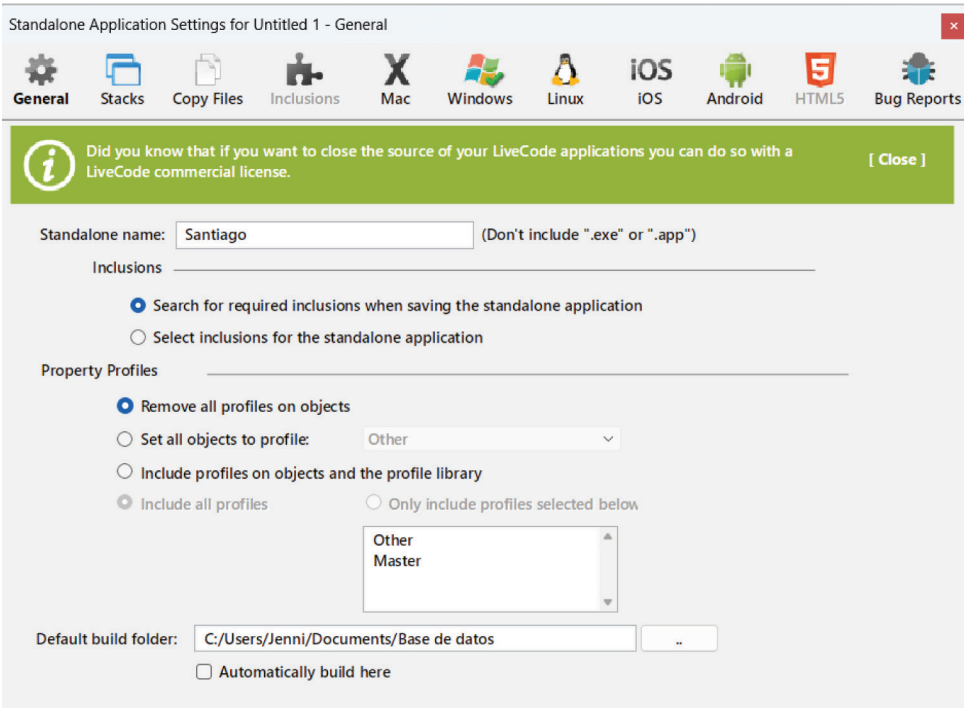


Fig. 2.32

Como se observa, en la figura 2.32 están presentes los sistemas operativos Mac, Windows, Linux, Android y el formato Web HTML5.

Antes de definir el sistema operativo, en el cual el usuario desea compilar, este debe nombrar la aplicación. En el caso presentado, la aplicación se denomina "Santiago".

Es importante observar que en dependencia de las bibliotecas especializadas que este use, será necesario seleccionar elementos de la opción ***Inclusions,*** esto se refiere, por ejemplo, al empleo de bases de datos no nativas, cursores, navegadores, motor de animaciones, etcétera.

Si este no ha empleado ninguna función o método de estas bibliotecas, podrá dejar marcada la opción implícita. Acto seguido deberá pasar a la ficha que se corresponde con el sistema operativo para el cual desea realizar su ejecutable. A continuación se verán algunos ejemplos:

Compilar para Windows (figura 2.33):

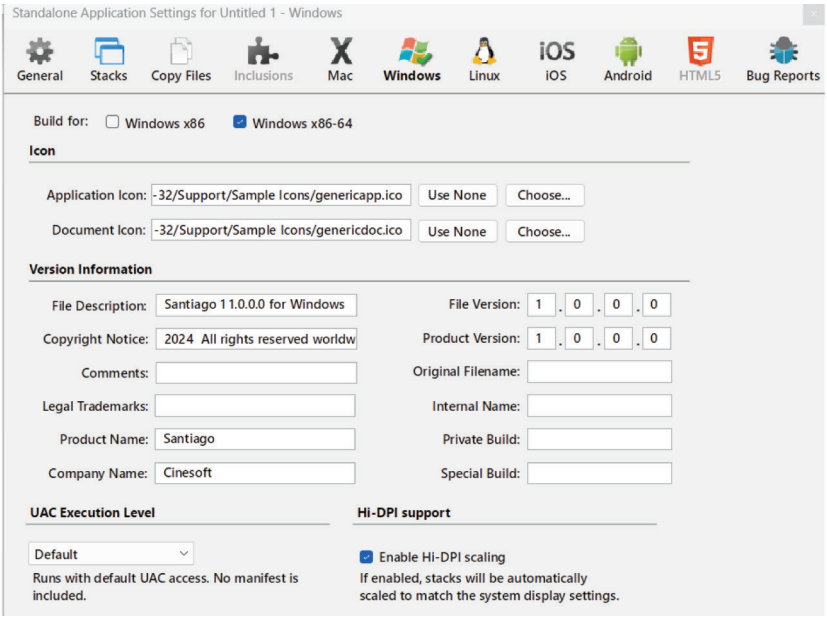


Fig. 2.33

En este cuadro de diálogo el usuario puede seleccionar un ícono: (***Application icon*** y ***Document icon***, que eventualmente pueden ser el mismo ícono y completar el campo ***File description***, asignándole un identificador con número de versión a su aplicación. Puede completar los datos de ***Product name*** y el nombre de la compañía o institución a la cual pertenece. De esta forma quedará configurada la salida para Windows.

Compilar para Android:

En el caso de la configuración para Android, se deberán tener en cuenta otros requisitos, como los siguientes:

- Lograr la adaptación a las dimensiones del dispositivo (teléfono, tableta, etcétera) y tener en cuenta si se emplea algún dispositivo especial del teléfono (cámara, vibración, etcétera).
- Instalar previamente el ***JDK de Java.***
- Instalar y configurar el ***SDK de Android.***

Estas dos bibliotecas son libres y se pueden descargar fácilmente desde Internet o solicitársela a un compañero.

Con relación al ***JDK de Java,*** será necesario solamente instalarlo, ya que Livecode lo encontrará de manera automática; sin embargo, el ***SDK de Android*** deberá ser instalado mediante la siguiente secuencia:

Menú ***Edit- Preferences – Mobil support,*** y buscar el camino en que se encuentra copiado el ***SDK de Android*** (figura 2.34).

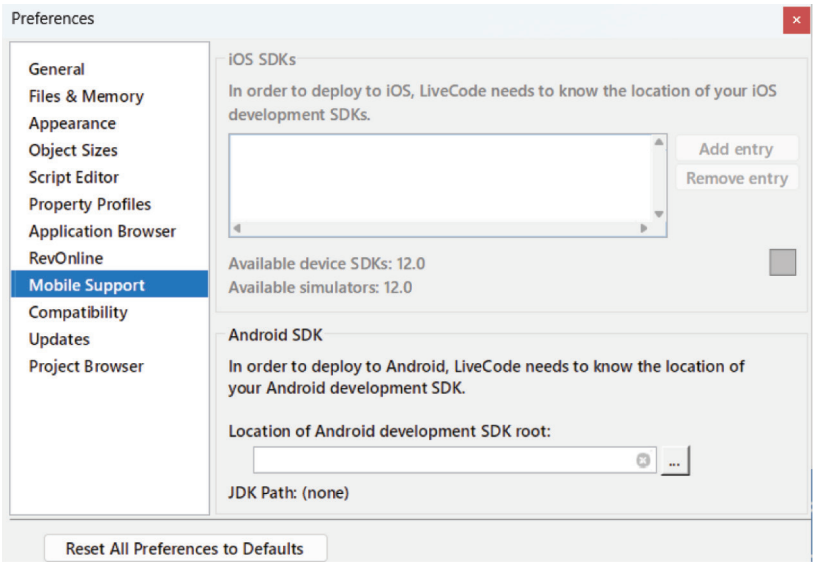


Fig. 2.34

Con esto Livecode estará listo para compilar para Android; por tanto, se podrá pasar a configurar la salida de igual manera que se hizo con Windows.

Esta vez deberá completar los campos ***Label*** con el nombre de la aplicación, puede completar la identificación del ***apk*** con tres atributos separados por punto, por ejemplo: ***com.nombre de la institución.nombre del apk*** (figura 2.35).

Lo más importante es asignar el valor ***Sing for development only*** al campo ***Signing.***

Finalmente, se deberá tener en cuenta la orientación inicial del móvil, que puede asumir dos valores: ***Initial orientation: Landscape*** (horizontal o apaisado) y ***Portariat*** (vertical o retrato).

Posterior a la configuración, se pasa al segundo paso, o sea, a la generación del ejecutable (***Save as Stand Alone Application***) del menú ***File.***

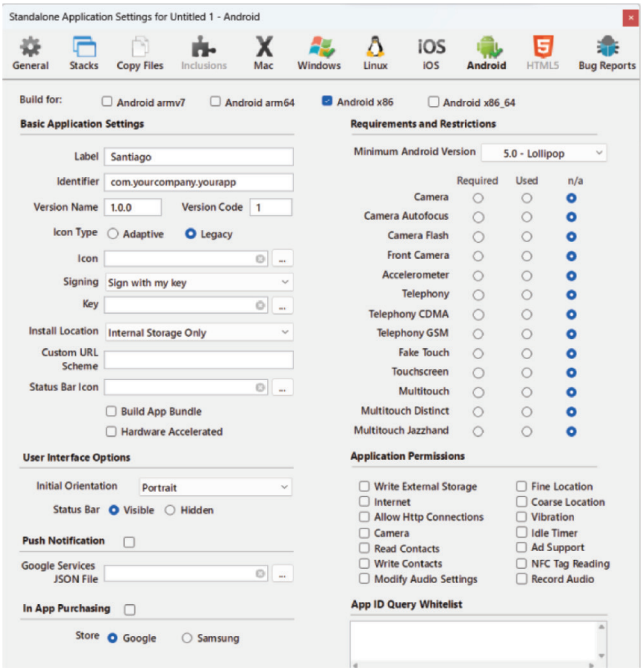


Fig. 2.35

Ante esta opción, Livecode solicitará una carpeta en la que depositará la aplicación compilada ***cojn*** extensión ***.exe,*** si se trata de Windows, o con la extensión .***apk*** si se trata de Android.

Nota: si su desarrollo es para móviles, es conveniente tener en cuenta el ajuste de la aplicación a las dimensiones del dispositivo. Para esto puede incluir un código en el mensaje ***preOpenStack*** en el ***script*** del ***stack*** como sigue:

```
on preOpenStack
   if the environment is "mobile" then
      set the fullscreen of this stack to true
      set the fullscreenmode of this stack to "exactFit"
   else
      set the fullscreen of this stack to true
      set the fullscreenmode of this stack to "letterbox"
   end if
end preOpenStack
```

Comprueba lo aprendido

1. ¿En qué elementos radica la dificultad de resolver problemas?
2. ¿Qué tres momentos son invariantes en la solución de problemas?
3. ¿A qué se llama propiedades de objetos y sobre qué estas pueden influir?
4. ¿De cuántas maneras se pueden modificar las propiedades de un objeto usando LiveCode?
5. Menciona algunas secciones del cuadro de diálogo de propiedades de objetos en LiveCode
6. ¿Cuál es la sintaxis general para asignar un valor a una propiedad?
7. ¿Qué es una propiedad de usuario y cómo se define?
8. ¿Cuál es la importancia de las propiedades de usuario?
9. ¿Qué es una Lista en LiveCode?
10. ¿Cómo se logra obtener un elemento de una lista en LiveCode?
11. ¿A qué se llama Evento y qué diferencia o similitud tiene este concepto con el concepto Mensaje?
12. Menciona cinco eventos y sus respectivos mensajes manejables por LiveCode.
13. Explica con tus palabras la jerarquía de mensajes de LiveCode.
14. En qué nivel de jerarquía se debe colocar un manipulador de mensajes.
15. ¿Cuál es la sintaxis para definir un mensaje personalizado o un mensaje definido por el programador?

16. ¿Qué relación guardan los conceptos de Función en Matemática y en Informática?

17. Menciona tres funciones ***build-in*** de LiveCode.

18. ¿Cuál es la sintaxis para definir una función definida por el programador?

19. ¿A qué se llama variable global y cómo se define?

20. ¿Cómo se pueden almacenar varios datos en una misma variable?

21. ¿Cómo se nombran los arreglos de una dimensión?

22. ¿Cómo se nombran los arreglos de dos dimensiones?

23. ¿Qué es un arreglo de listas?

24. ¿En qué contextos es pertinente el uso de ficheros?

25. Resuelve los siguientes problemas, codificándolos en LiveCode:

a) Escribe un programa que convierta de centímetros a pulgadas. Una pulgada es igual a 2,54 cm.
b) Escribe un programa que determine si el número entero ingresado por el usuario es par o no.
c) Escribe un programa que pida dos números enteros y que calcule la división, indicando si la división es exacta o no.
d) Escribe un programa que pida al usuario dos palabras, y que indique cuál de ellas es la más larga y por cuántas letras lo es.
e) Escribe un programa que reciba como entrada dos números, y los muestre ordenados de menor a mayor.
f) Escribe un programa que simule una calculadora básica, este puede realizar operación de suma, resta, multiplicación y división. El programa debe recibir como entrada dos números reales y un operador, que puede ser +, -, * o /. La salida del programa debe ser el resultado de la operación.
g) Escribe un programa que entregue el año, mes, día de nacimiento, la edad y el sexo del usuario a partir de su carnet de identidad.

h) Escribe un programa que pida al usuario dos números enteros, y luego entregue la suma de todos los números que están entre ellos. Por ejemplo, si los números son 1 y 7, debe entregar como resultado 2 + 3 + 4 + 5 + 6 = 20.
i) Escribe un programa que pida al usuario un entero de tres dígitos, y entregue el número con los dígitos en orden inverso.
j) En una fábrica se controla cada mes la asistencia de sus trabajadores. Se conoce la cantidad de trabajadores que asistieron en cada uno de los tres primeros meses del año y se desea calcular el promedio mensual de asistencia de los trabajadores en el trimestre.
k) Escribe un programa que determine si un número entero positivo dado, es primo o no.
l) Escribe un programa que, dada una sucesión de valores numéricos, genere una nueva sucesión cuyos elementos sean los de la colección original pero en orden inverso.
m) Escribe un programa que, dada una lista de elementos devuelva la lista sin valores repetidos (alizar la lista).
n) Dada una sucesión numérica, genere otra colección con los valores positivos de la colección original y calcule el producto de ellos.
o) Escribe un programa que determine el menor de los valores de una sucesión de números, de la que se conoce la cantidad de elementos, así como la posición que ocupa ese valor en la sucesión.
p) Implementa las funciones de la teoría de conjuntos para dos listas de elementos.

26. Desarrolla una presentación con diapositivas al estilo ***PowerPoint*** o ***Impres*** usando Livecode.

27. Elabora una animación de un avión y un autobús que siguen diferentes trayectorias viajando desde Santiago de Cuba a La Habana.

28. Desarrolla el juego del ahorcado usando Livecode y compílalo para ***Android.***

29. Desarrolla una calculadora para ***Android.***

30. Desarrolla una galería de fotos para ***Android.***

31. Desarrolla una base de datos con el directorio de los educandos de tu aula para ***Windows*** y para ***Android.***

32. Desarrolla un ***apk*** con el calendario escolar.

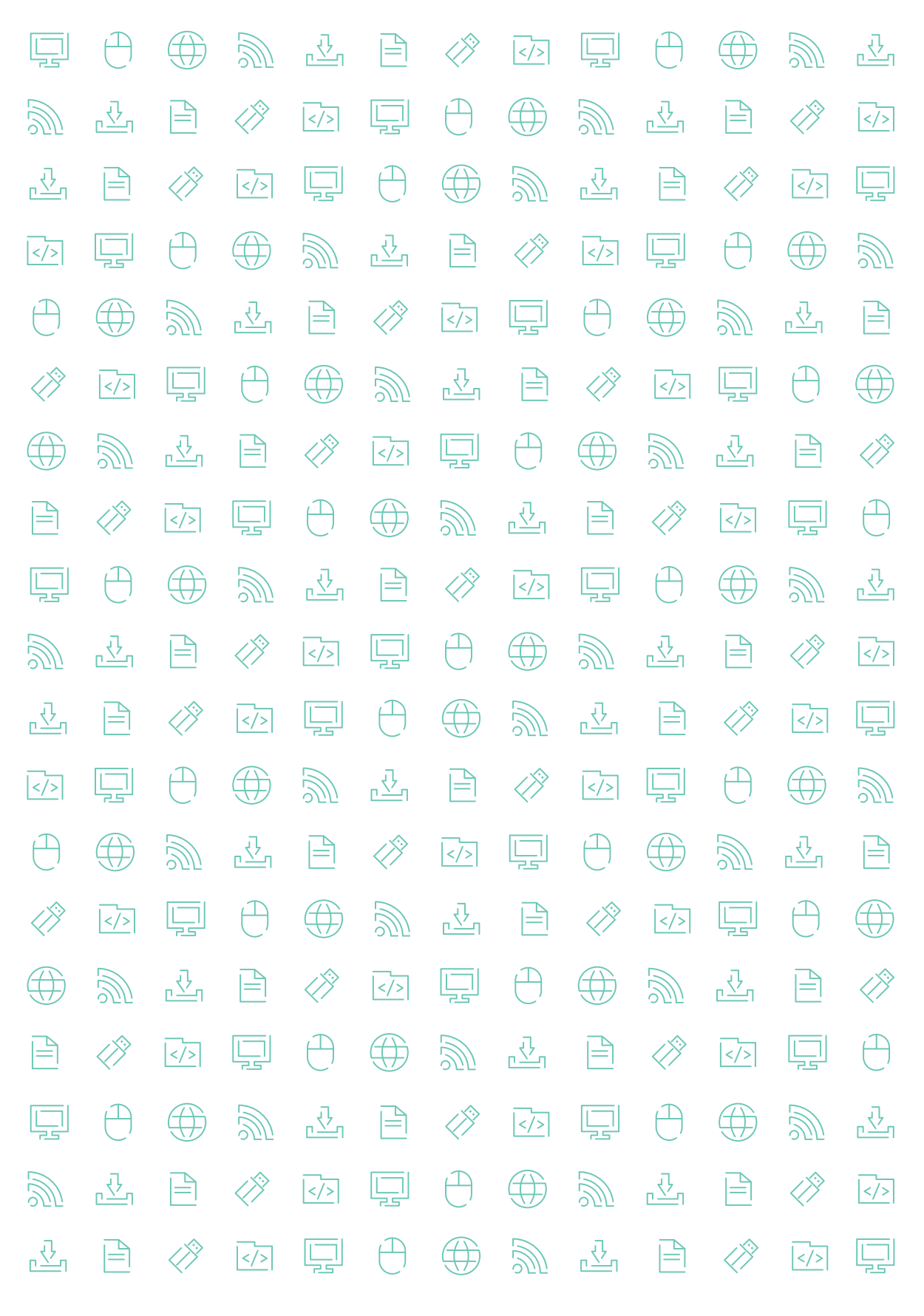
33. Desarrolla una revista de tu escuela.

34. Elabora un libro electrónico sobre carreras universitarias.

35. Determina la cantidad de números entre 100 y 999 con cifras diferentes cuya suma de las cifras sea igual a un número dado.

36. Elabora un programa que reciba el carnet de identidad de una persona y diga:

a) Si es menor de edad, adolescente o adulto.
b) Si es de sexo masculino o femenino.
c) Cuál es su signo zodiacal.



BIBLIOGRAFÍA

Basogain Olabe, X.: "Pensamiento computacional a través de la programación: paradigma de Aprendizaje", www.um.es/ead/red/46/Basogain. pdf, 2015

Budd, T.: ***Introducción a la programación orientada a objetos,*** Ed. Addison-Wesley Iberoamericana, Madrid, 1994.

Cangalaya Sevillano, J. A.: ***Estrategias de aprendizaje de la metodología activa***, Ed. Grupo de Capacitación Pedagógica, Lima, 2010.

Chun, B. y T. Piotrowski: "Pensamiento computacional ilustrado", www.cti-llustrated.com/

Colectivo de autores: "Pensamiento computacional. Un aporte para la educación de hoy", Montevideo, Uruguay, www.gurisesunidos.org.uy

Fundación Misión Sucre: ***Ejemplos y ejercicios. Algoritmia,*** Ministerio de de Educación Superior de la República Bolivariana de Venezuela, Venezuela, [s. a.].

García, M. E.: "Introducción a la Informática. Ejercicios resueltos de algoritmos", www.profmatiasgarcia.com.ar

Garriga Garzón, F.: ***Problemas resueltos de programación lineal,*** Ed. Omnia Publisher, versión digital.

Goldberg, S.: ***LiveCode Lite: Computer Programming Made Ridiculously Simple,*** Ed. MedMaster Inc, Miami, Fl., [s. a.].

Holgate, C. y J. Gerdeen: "LiveCode Mobile Development Beginner's Guide", (segunda), Ed. Packt Publishing Ltd, www.packtpub.com

Joyanes Aguilar, L.: ***Fundamentos de programación. Algoritmos y Estructuras de datos y objetos***, Ed. McGraw-Hill, Madrid, 2003.

Lavieri E.: "LiveCode Mobile Development Cookbook", primera parte, Ed. Packt Publishing Ltd, Birmingham–Mumbai, www.packtpub.com, 2014.

López Cruz, P. A.: ***Manual fundamental. Hardware y componentes,*** Ed. Anaya, [s. a.].

López García, J. C.: "Algoritmos y programación. Guía para docentes", www.eduteka.org, 2009.

Muñoz, A.: "La imagen fotográfica", Ministerio de Educación, Política Social y Deporte, Centro Nacional de Información y Comunicación Educativa, Madrid, en http://w3.cnice.mec.es/eos/MaterialesEducativos/ mem2006/ fotografi a/index.html#

Noguez Monroy, J.: ***Informática***, segunda parte, Ed. Santillana, México, 2006.

Osuna Guerrero, R. y E. García Cerro: "Fundamentos de fotografía digital", UNED, Madrid, http://ocw.innova.uned.es/ocwuniversia/tecnologia-electronica/fundamentos-de-fotografia-digital.

Pérez Narváez, H. O.: "Herramientas informáticas para el desarrollo del Pensamiento computacional", versión digital.

Quereda Gómez, E. D., y L. Bañón Blázquez: "Colección de exámenes de la asignatura Aplicación de Ordenadores 2005–2011. Fundamentos de programación y algoritmia", versión digital.

Rodríguez Rodríguez, L. A.: ***Revolución multimedia,*** ISP Felix Varela, Centro de estudios Visofted, [s. a.].

Sta Ilings, W.: ***Organización y arquitectura de computadores,*** séptima edición, Ed. Pearson Educación, S. A., Madrid, 2005.

Tamayo Silva , F.: ***Lógica de programación y lenguajes,*** Instituto Superior Pedagógico Pepito Tey, Las Tunas, 2002.

Zapata Ospina, C. A.: F***undamentos de programación. Guía de autoenseñanza,*** (primera), RA-MA editorial, Colombia, 2006.

Zavaleta Luna, D. M. y J. J. Francisco León: "LiveCode para Android a través de ejemplos: nivel básico, (primera)", Universidad Juárez Autónoma de Tabasco, México, www.pensamientocomputacional.org

EDITORIAL
PUEBLO Y EDUCACIÓN

9 789591 348173