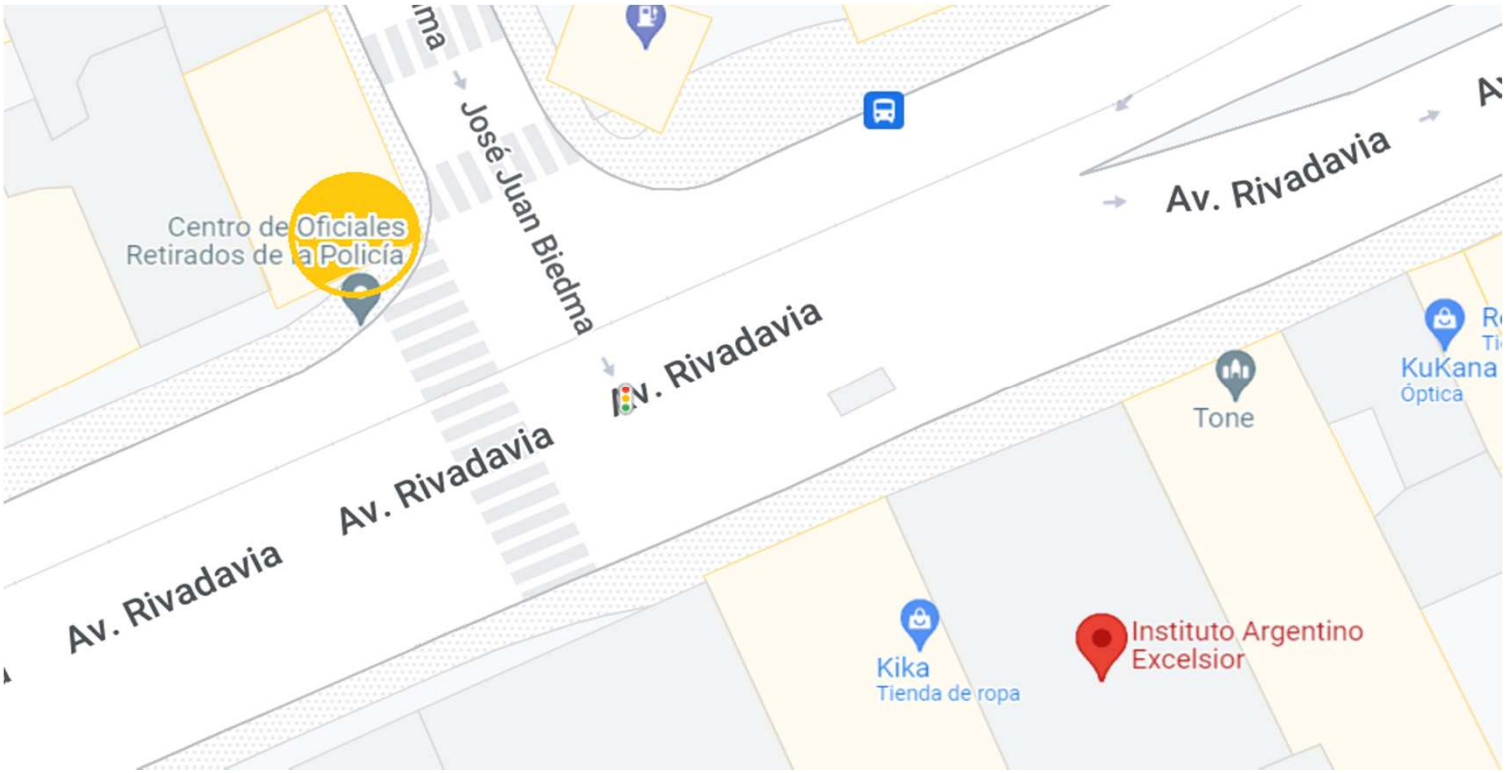
Concepto de algoritmo

Llamamos “algoritmo” al conjunto finito y ordenado de acciones con las que podemos resolver un determinado problema.

Llamamos “problema” a una situación que se nos presenta y que, mediante la aplicación de un algoritmo, pretendemos resolver.

Ejemplo

Llegamos a la intersección de la calle Biedma con la avenida Rivadavia, y pretendemos cruzar para llegar al instituto Excelsior.



Intuitivamente, aplicaremos el siguiente algoritmo:

- Esperar a que la luz del semáforo peatonal este en verde (o en blanco)(esperarSemaforo);
- Cruzar la calle (cruzarCalle);

Este algoritmo esta compuesto por las acciones:

- esperarSemaforo
- cruzarCalle;

La acción esperarSemaforo implica:

- Observar la luz del semaforo (observarLuz);
- **Si** la luz esta en “rojo” o en “verde intermitente” **entonces**

- esperar un momento (esperar);
- **ir a**: observarLuz;

La acción cruzarCalle implica:

- Bajar la calzada (bajarCalzada);
- Avanzar un paso (avanzarPaso);
- **Si** la distancia hacia la otra calzada es mayor que la distancia que podemos avanzar dando un nuevo paso **entonces**
- - **ir a**: avanzarPaso;
- Subir la calzada de la vereda de enfrente (subirCalzada);

Análisis del enunciado de un problema

- Comprender el alcance.
- Identificar los datos de entrada.
- Identificar los datos de salida o resultados.

Análisis del problema

Llegamos a una esquina y pretendemos cruzar la calle. Para poder cruzar con la mayor seguridad posible, tenemos que esperar que el semáforo habilite el paso peatonal. Hasta que esto ocurra nos encontraremos detenidos al lado del semáforo y una vez que este lo permita avanzaremos, paso a paso, hasta llegar a la vereda de enfrente. Se considera que el semáforo habilita el paso peatonal cuando emite una luz “verde fija”. En cambio si el semáforo emite una luz “roja” o “verde intermitente” se recomienda esperar.

Datos de entrada

Podemos identificar los siguientes datos de entrada:

- Nuestra posición inicial - posición en donde nos detuvimos a esperar a que el semáforo habilite el paso peatonal. La llamaremos posInicial.
- Luz del semáforo - el color de la luz que el semáforo está emitiendo. Lo llamaremos luz.
- Distancia de avance de paso - que distancia avanzamos cada vez que damos un nuevo paso. La llamaremos distPaso.
- Posición de la vereda de enfrente. La llamaremos posFinal ya que es allí hacia donde nos dirigimos.

Datos de salida

Si bien en este problema no se identifican datos de salida resulta evidente que luego de ejecutar la secuencia de acciones del algoritmo nuestra posición actual deberá ser la misma que la posición de la vereda de enfrente. Es decir, si llamamos p a nuestra posición actual resultará que su valor al inicio del algoritmo será posInicial y al final del mismo deberá ser posFinal.

Memoria y operaciones aritméticas y lógicas

En la vida real, cuando observamos la luz del semáforo almacenamos este dato en algún lugar de nuestra memoria para poder evaluar si corresponde cruzar la calle o no. Generalmente, los datos de entrada ingresan al algoritmo a través de una acción que llamamos “lectura” o “ingreso de datos” y permanecen en la memoria el tiempo durante el cual el algoritmo se esta ejecutando. Por otro lado, el hecho de evaluar si corresponde o no cruzar la calle implica realizar una operación lógica. Esto es: determinar si la proposición “el semáforo emite luz roja o verde intermitente” resulta ser verdadera o falsa. También realizaremos una operación lógica cuando estemos cruzando la calle y tengamos que determinar si corresponde dar un nuevo paso o no. Recordemos que llamamos p a nuestra posición actual, posFinal a la posición de la vereda de enfrente y distPaso a la distancia que avanzamos dando un nuevo paso. Por lo tanto, corresponde dar otro paso si se verifica que posFinal > p+distPaso. Aquí además de la operación lógica estamos realizando una operación aritmética: la suma p+distPaso.
En resumen, estos son los recursos que tenemos disponibles para diseñar y desarrollar algoritmos:

- La memoria.
- La posibilidad de realizar operaciones aritméticas.
- La posibilidad de realizar operaciones lógicas.

Teorema de la programación estructurada

Este teorema establece que todo algoritmo puede resolverse mediante el uso de tres estructuras básicas llamadas “estructuras de control”:

- La “estructura secuencial” o “acción simple”.
- La “estructura de decisión” o “acción condicional”.
- La “estructura iterativa” o “acción de repetición”.

Dos de estas estructuras las hemos utilizado en nuestro ejemplo de cruzar la calle. La estructura secuencial es la más sencilla y simplemente implica ejecutar una acción, luego otra y así sucesivamente. La estructura de decisión la utilizamos para evaluar si correspondía dar un nuevo paso en función de nuestra ubicación actual y la ubicación de la vereda de enfrente. Sin embargo, para resolver el problema hemos utilizado una estructura tabu: la estructura “**ir a**” o, en ingles, “*go to*” o “*goto*”. Esta estructura quedo descartada luego de que el teorema de la programación estructurada demostrara que con una adecuada combinación de las tres estructuras de control antes mencionadas es posible resolver cualquier algoritmo sin tener que recurrir al “*goto*” (o estructura “ir a”). Para ejemplificarlo vamos a replantear el desarrollo de los algoritmos anteriores y reemplazaremos la estructura “**ir a**” (*goto*) por una estructura iterativa: la estructura “**repetir mientras que**”.

esperarSemaforo

Con "**ir a**"	Sin "**ir a**"
• observarLuz; • **Si** la luz esta en "rojo" o en "verde intermitente" **entonces** - esperar; - **ir a**: observarLuz;	• observarLuz; • **Repetir mientras que** la luz este en "rojo" o en "verde intermitente" **hacer** - esperar; - observarLuz;

cruzarCalle

Con "**ir a**"	Sin "**ir a**"
• bajarCalzada; • avanzarPaso; • **Si** la distancia hacia la otra calzada es mayor que la distancia que podemos avanzar dando un nuevo paso **entonces** - **ir a**: avanzarPaso; • subirCalzada;	• bajarCalzada; • avanzarPaso; • **Repetir mientras que** la distancia hacia la otra calzada sea mayor que la distancia que podemos avanzar dando un nuevo paso **hacer** - avanzarPaso; • subirCalzada;

Como vemos, la combinación “acción condicional + *goto*” se puede reemplazar por una estructura de repetición. Si bien ambos desarrollos son equivalentes la nueva versión es mejor porque evita el uso del *goto*, estructura que quedo en desuso porque trae grandes problemas de mantenibilidad.

Conceptos de programación

En general, estudiamos algoritmos para aplicarlos a la resolución de problemas mediante el uso de la computadora. Las computadoras tienen memoria y tienen la capacidad de resolver operaciones aritméticas y lógicas. Por lo tanto, son la herramienta fundamental para ejecutar los algoritmos que vamos a desarrollar. Para que una computadora pueda ejecutar las acciones que componen un algoritmo tendremos que especificarlas o describirlas de forma tal que las pueda comprender. Todos escuchamos alguna vez que “las computadoras solo entienden 1 (unos) y 0 (ceros)” y, efectivamente, esto es así, pero para nosotros (simples humanos) especificar las acciones de nuestros algoritmos como diferentes combinaciones de unos y ceros seria una tarea verdaderamente difícil. Afortunadamente, existen los lenguajes de programación que proveen una solución a este problema.

Lenguajes de programación

Las computadoras entienden el lenguaje binario (unos y ceros) y nosotros, los humanos, entendemos lenguajes naturales (español, ingles, portugués, etc.).
Los lenguajes de programación son lenguajes formales que se componen de un conjunto de palabras, generalmente en ingles, y reglas sintácticas y semánticas.
Podemos utilizar un lenguaje de programación para escribir o codificar nuestro algoritmo y luego, con un programa especial llamado “compilador”, podremos generar los “unos y ceros” que representan sus acciones. De esta manera, la computadora será capaz de comprender y convertir al algoritmo en un programa de computación.
Existen muchos lenguajes de programación: Pascal, C, Java, COBOL, Basic, Smalltalk, etc., y también existen muchos lenguajes derivados de los anteriores: Delphi (derivado de Pascal), C++ (derivado de C), C# (derivado de C++), Visual Basic (derivado de Basic), etcétera.
Los lenguajes naturales son los que hablamos y escribimos los seres humanos: inglés, español, italiano, etc. Son dinámicos ya que, constantemente, incorporan nuevas variaciones, palabras y significados. Por el contrario, los lenguajes formales son rígidos y se construyen a partir de un conjunto de símbolos (alfabeto) unidos por un conjunto de reglas (gramática). Los lenguajes de programación son lenguajes formales.

Lenguajes de programación - Ejemplos

El lenguaje de programación C fue creado por Dennis Ritchie (1941-2011).
En 1967 Ritchie comenzó a trabajar para los laboratorios Bell, donde se ocupó, entre otros, del desarrollo del lenguaje B. Creó el lenguaje de programación C en 1972, junto con Ken Thompson.
Ritchie también participó en el desarrollo del sistema operativo Unix.
El lenguaje C++ fue creado a mediados de los años ochenta por Bjarne Stroustrup, con el objetivo de extender al lenguaje de programación C con mecanismos que permitan la manipulación de objetos.
Java es un lenguaje de programación orientado a objetos, creado en la década del noventa por Sun Microsystems (actualmente adquirida por Oracle).
Utiliza gran parte de la sintaxis de C y C++, pero su modelo de objetos es más simple.

Codificación de un algoritmo

Cuando escribimos las acciones de un algoritmo en algún lenguaje de programación decimos que lo estamos "codificando". Generalmente, cada acción se codifica en una línea de código. Al conjunto de líneas de código, que obtenemos luego de codificar el algoritmo, lo llamamos "código fuente". El código fuente debe estar contenido en un archivo de texto cuyo nombre debe tener una extensión determinada que dependerá del lenguaje de programación que hayamos utilizado. Por ejemplo, si codificamos en Pascal entonces el nombre del archivo debe tener la extensión ".pas". Si codificamos en Java entonces la extensión del nombre del archivo deberá ser ".java" y si el algoritmo fue codificado en C entonces el nombre del archivo deberá tener la extensión ".c".

Bibliotecas de funciones

Los lenguajes de programación proveen bibliotecas o conjuntos de funciones a través de las cuales se ofrece cierta funcionalidad básica que permite, por ejemplo, leer y escribir datos sobre cualquier dispositivo de entrada/salida como podría ser la consola.
Es decir, gracias a que los lenguajes proveen estos conjuntos de funciones los programadores están exentos de programarlas y simplemente se limitan a utilizarlas.
Programando en C, por ejemplo, cuando se necesite mostrar un mensaje en la pantalla se utilizará la función printf y cuando se desee leer datos a través del teclado se utilizará la función scanf. Ambas funciones forman parte de la biblioteca estándar de entrada/salida de C, también conocida como "stdio.h".

Programas de computación

Un programa es un algoritmo que ha sido codificado y compilado y que, por lo tanto, puede ser ejecutado en una computadora.
El algoritmo constituye la lógica del programa. El programa se limita a ejecutar cada una de las acciones del algoritmo. Por esto, si el algoritmo tiene algún error entonces el programa también lo tendrá y si el algoritmo es lógicamente perfecto entonces el programa también lo será.
El proceso para crear un nuevo programa es el siguiente:

- Diseñar y desarrollar su algoritmo.
- Codificar el algoritmo utilizando un lenguaje de programación.
- Compilarlo para obtener el código de maquina o archivo ejecutable (los “unos y ceros”).

Dado que el algoritmo es la parte lógica del programa muchas veces se utilizan ambos términos como sinónimos ya que para hacer un programa primero se necesita diseñar su algoritmo. Por otro lado, si se desarrolla un algoritmo seguramente será para, luego, codificarlo y compilarlo, es decir, programarlo.

Consola - Entrada y salida de datos

Llamamos “consola” al conjunto compuesto por el teclado y la pantalla de la computadora en modo texto. Cuando se hablemos de ingreso de datos por consola nos estaremos refiriendo al teclado y cuando hablemos de mostrar datos por consola estaremos hablando de la pantalla, siempre en modo texto.
Llamamos “entrada” al conjunto de datos externos que ingresan al algoritmo. Por ejemplo, el ingreso de datos por teclado, la información que se lee a través de algún dispositivo como podría ser un lector de código de barras, un *scanner* de huellas digitales, etcétera.
Llamamos “salida” a la información que el algoritmo emite sobre algún dispositivo como ser la consola, una impresora, un archivo, etcétera.
La consola es el dispositivo de entrada y salida por omisión. Es decir, si hablamos de ingreso de datos y no especificamos nada más será porque el ingreso de datos lo haremos a través del teclado. Análogamente, si hablamos de mostrar cierta información y no especificamos mas detalles nos estaremos refiriendo a mostrarla por la pantalla de la computadora, en modo texto.

Lenguajes algorítmicos

Llamamos "lenguaje algorítmico" a todo recurso que permita describir con mayor o menor nivel de detalle los pasos que componen un algoritmo.
Los lenguajes de programación, por ejemplo, son lenguajes algorítmicos ya que la codificación del algoritmo es en si misma una descripción detallada de los pasos que lo componen. Sin embargo, para describir un algoritmo no siempre será necesario llegar al nivel de detalle que implica codificarlo. Una opción valida es utilizar un "pseudocódigo".

Pseudocódigo

El pseudocódigo surge de mezclar un lenguaje natural (por ejemplo, el español) con ciertas convenciones sintácticas y semánticas propias de un lenguaje de programación. Justamente, el algoritmo que resuelve el problema de "cruzar la calle" fue desarrollado utilizando un pseudocodigo.
Los diagramas también permiten detallar los pasos que componen un algoritmo; por lo tanto, son lenguajes algorítmicos.

Representación gráfica de algoritmos

Los algoritmos pueden representarse mediante el uso de diagramas. Los diagramas proveen una visión simplificada de la lógica del algoritmo y son una herramienta importantísima que se utilizan para analizar y documentar los algoritmos o programas a desarrollar.
Como ejemplo se citan los llamados diagramas de *Nassi-Shneiderman,* también conocidos como "diagramas de Chapin".
Estos diagramas se componen de un conjunto de símbolos que permiten representar cada una de las estructuras de control que describe el teorema de la programación estructurada: la estructura secuencial, la estructura de decisión y la estructura de repetición.
Los diagramas Nassi-Shneiderman, fueron publicados en 1973 por Ben Shneiderman e Isaac Nassi en el artículo llamado "A short history of strutctured flowcharts", con el objetivo de eliminar las líneas de los diagramas tradicionales y así reforzar la idea de estructuras de “única entrada y única salida”.
Si en la programación estructurada se elimina la sentencia GOTO entonces se deben eliminar las líneas en los diagramas que la representan.