Unidad 1:

Computadora y Programación: conceptos básicos.

Equipo de Cátedra
Elementos de Programación

La programación y el tratamiento digital de la información son pilares fundamentales en la formación de todo profesional vinculado al análisis y la gestión de datos. Esta unidad tiene como propósito introducir los conceptos esenciales sobre el funcionamiento de una computadora, el rol del software y del hardware y los lenguajes de programación.

1. ¿QUÉ ES UNA COMPUTADORA?

1.1. Componentes de una computadora.

Las computadoras son una parte esencial de nuestra vida cotidiana. Casi todos los aparatos que usamos tienen algún tipo de computadora capaz de ejecutar ciertas tareas: lavarropas con distintos modos de lavado, consolas de juegos para momentos de entretenimiento, calculadoras súper potentes, computadoras personales que se usan para un montón de propósitos, teléfonos celulares con un sinfín de aplicaciones y miles de cosas más.

Todos estos dispositivos con computadoras de distinto tipo tienen algo en común: alguien "les dice" cómo funcionar, es decir, les indica cuáles son los pasos que deben seguir para cumplir una tarea. De eso se trata la programación: es la actividad mediante la cual las personas le entregan a una computadora un conjunto de instrucciones para que, al ejecutarlas, ésta pueda resolver un problema. Quienes realizan esta actividad reciben el nombre de programadores. Sin las personas que las programan, las computadoras dejan de ser útiles, por más complejos que sean estos aparatos. Los conjuntos de instrucciones que reciben las computadoras reciben el nombre de programas.

La programación es un proceso creativo: en muchas ocasiones la tarea en cuestión puede cumplirse siguiendo distintos caminos y el programador es el que debe imaginar cuáles son y elegir uno. Algunos de estos caminos pueden ser mejores que otros, pero en cualquier caso la computadora se limitará a seguir las instrucciones ideadas por el programador.

Desafortunadamente, las computadoras no entienden español ni otro idioma humano. Hay que pasarles las instrucciones en un lenguaje que sean capaces de entender. Para eso debemos aprender algún lenguaje de programación, que no es más que un lenguaje artificial compuesto por una serie de expresiones que la computadora puede interpretar. Las computadoras interpretan nuestras instrucciones de forma muy literal, por lo tanto, a la hora de programar hay que ser muy específicos. Es necesario respetar las reglas del lenguaje de programación y ser claros en las indicaciones provistas.

2. HARDWARE Y SOFTWARE

En la historia de la computación hubo dos aspectos que fueron evolucionando: las máquinas y los programas que las dirigen. Hacemos referencia a estos elementos como hardware y software respectivamente, y es la conjunción de ambos la que le da vida a la computación y hace posible la programación.

Figura 1.1: Representación de la diferencia entre hardware y software.

El hardware es el conjunto de piezas físicas y tangibles de la computadora. Existen diversas formas de clasificar a los elementos que componen al hardware, según distintos criterios:

Tabla 1.1: Clasificación del hardware

Criterio	Clasificación	Descripción	Ejemplos
Según su utilidad	Dispositivos de procesamiento	Son los que reciben las instrucciones mediante señales eléctricas y usan cálculos y lógica para interpretarlas y emitir otras señales eléctricas como resultado.	microprocesador, tarjeta gráfica, tarjeta de sonido, etc.
	Dispositivos de almacenamiento	Son capaces de guardar información para que esté disponible para el sistema.	disco duro, pen drive, DVD, etc.
	Dispositivos de entrada	Captan instrucciones por parte de los usuarios y las transforman en señales eléctricas interpretables por la máquina.	teclado, mouse, touch pad, etc.
	Dispositivos de salida	Transforman los resultados de los dispositivos de procesamiento para presentarlos de una forma fácilmente interpretable para el usuario.	monitor, impresora, etc.

Según su ubicación	Dispositivos internos	Generalmente se incluye dentro de la carcasa de la computadora.	microprocesador, disco rígido, ventiladores, módem, tarjeta gráfica, fuente de alimentación, puertos, etc.
	Dispositivos externos o periféricos	No se incluye dentro de la carcasa de la computadora y está al alcance del usuario	Monitor, teclado, mouse, joystick, micrófono, impresora, escáner, pen drive, lectores de código de barras, etc.
Según su importancia	Hardware principal	Dispositivos esenciales para el funcionamiento de la computadora	microprocesador, disco rígido, memoria RAM, fuente de alimentación, monitor, etc.
	Hardware complementario	Aquellos elementos no indispensables (claramente, dependiendo del contexto, alguna pieza del hardware que en alguna situación podría considerarse complementaria, en otras resulta principal).	

Por otro lado, tenemos al software, que es el conjunto de todos los programas (es decir, todas las instrucciones que recibe la computadora) que permiten que el hardware funcione y que se pueda concretar la ejecución de las tareas. No tiene una existencia física, sino que es intangible.

El software se puede clasificar de la siguiente forma:

Tabla 1.2: Clasificación del software

Clasificación	Descripción	Ejemplos

Software de sistema o software base	Son los programas informáticos que están escritos en lenguaje de bajo nivel como el de máquina o ensamblador y cuyas instrucciones controlan de forma directa el hardware.	BIOS (sistemas que se encargan de operaciones básicas como el arranque del sistema, la configuración del hardware, etc), sistemas operativos (Linux, Windows, iOS, Android), controladores o *drivers*, etc.
Software de aplicación o utilitario	Son los programas o aplicaciones que usamos habitualmente para realizar alguna tarea específica.	procesadores de texto como Word, reproductores de música, Whatsapp, navegadores web, juegos, etc.
Software de programación o de desarrollo	Son los programas y entornos que nos permiten desarrollar nuestras propias herramientas de software o nuevos programas. Aquí se incluyen los lenguajes de programación.	Python, C++, Java, R, etc.

3. UNIDADES DE MEDIDA DE ALMACENAMIENTO DE LA INFORMACIÓN

En informática, la información se representa y almacena en forma de bits, que son la unidad más pequeña de datos. Cada bit puede tener solo dos valores posibles: 0 o 1.

Para poder trabajar con cantidades más grandes de información, se agrupan los bits en unidades mayores. A continuación, se presentan las principales unidades de medida utilizadas en computación:

Tabla 1.3: Unidades de medida de almacenamiento de la información

Unidad	**Equivalencia**
Bit	Puede valer 0 o 1
Byte	8 bits
Kilobyte (KB)	1024 bytes
Megabyte (MB)	1024 KB
Gigabyte (GB)	1024 MB
Terabyte (TB)	1024 GB

Ejemplos prácticos:

- Un archivo de texto simple puede pesar unos pocos **KB**.
- Una canción en formato MP3 puede ocupar entre **3 y 5 MB**.
- Una película en alta definición puede ocupar **2 a 4 GB**.
- Un disco rígido puede tener capacidad de **500 GB** o más.

Es importante conocer estas unidades porque nos permite:

- Medir el tamaño de archivos y datos.
- Comparar capacidades de almacenamiento.
- Entender los límites de memoria y velocidad en distintos dispositivos.

4. LENGUAJES DE PROGRAMACIÓN.

4.1. Reseña histórica sobre la programación.

La historia de la programación está profundamente ligada al desarrollo de la computación. El término “computación” proviene del latín computare, que significa contar o calcular. En un comienzo, esta actividad era realizada por personas utilizando herramientas como el ábaco. Sin embargo, desde el siglo XVII comenzaron a surgir los primeros dispositivos mecánicos diseñados para automatizar cálculos matemáticos.

Uno de los pioneros fue John Napier, quien en 1617 desarrolló los huesos de Napier (ver Figura 4.1), un sistema de barras que facilitaba multiplicaciones y raíces. Poco después, en 1623, Wilhelm Schickard creó el reloj calculador, una máquina capaz de realizar sumas y restas de forma mecánica. A este invento le siguieron otros como la pascalina de Blaise Pascal (1645) y las calculadoras de Leibniz, todas basadas en engranajes y ruedas.

Figura 4.1: De izquierda a derecha: los huesos de Napier (Museo Arqueológico Nacional de España), el reloj calculador de Schickard (Museo de la Ciencia de la Universidad Pública de Navarra) y una pascalina del año 1952

El gran salto conceptual ocurrió en el siglo XIX. En 1801, Joseph Jacquard diseñó un telar que usaba tarjetas perforadas para automatizar patrones de tejido (ver Figura 4.2). Esta idea fue clave para el desarrollo posterior de las computadoras. Inspirado en ese concepto, el matemático Charles Babbage propuso en 1837 su máquina analítica, la primera idea de una computadora programable. Aunque nunca se construyó por completo, su diseño incorporaba tarjetas perforadas y podía ejecutar diferentes cálculos según el patrón de perforación. Por esta razón, Babbage es conocido como el padre de la computación.

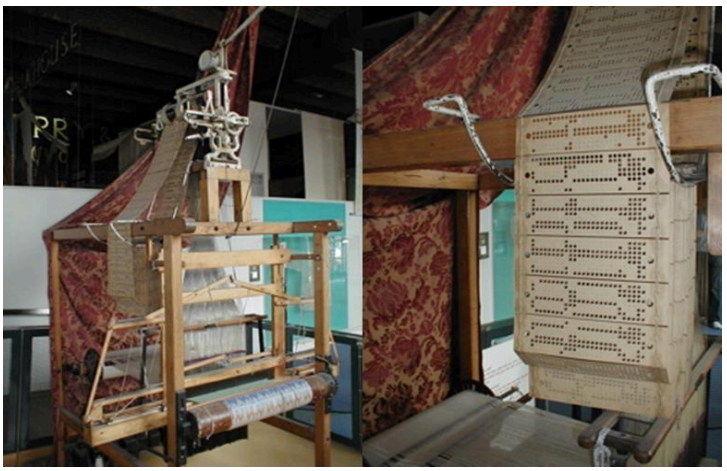

Figura 4.2: Un telar de Jacquard y sus tarjetas perforadas en el Museo de la Ciencia y la Industria en Mánchester.

La colaboración de Ada Lovelace, matemática británica, fue fundamental: en 1843 publicó notas sobre la máquina de Babbage, incluyendo el primer algoritmo para ejecutarse en una máquina. Su trabajo (ver Figura 4.3) la convierte en la primera programadora de la historia.

Figura 4.3: Charles Babbage, Ada Lovelace y el algoritmo que publicó Ada para calcular los números de Bernoulli con la máquina analítica de Charles.

A fines del siglo XIX, Herman Hollerith utilizó tarjetas perforadas para procesar el censo de EE.UU. en 1890, reduciendo el tiempo de procesamiento de 8 a 3 años. Fundó una empresa que luego se convertiría en IBM.

Ya en el siglo XX, con el avance de la electrónica, se desarrollaron computadoras como la ABC de Atanasoff y Berry (1942), y la ENIAC (1945), la primera computadora electrónica de propósito general. Aunque estas máquinas eran enormes y complejas, abrieron el camino para el diseño de computadoras más flexibles. La ENIAC, por ejemplo, usaba más de 18.000 tubos de vacío y ocupaba una sala completa (Figura 4.4).

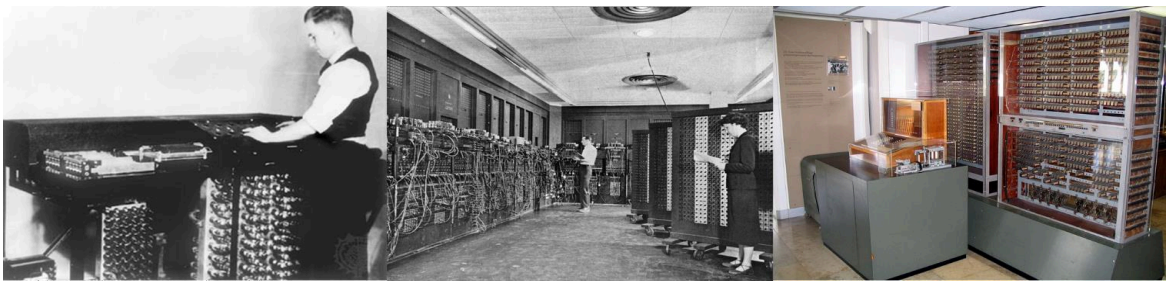

Figura 4.4: De izquierda a derecha: las computadoras ABC, ENIAC y Z3

El matemático John von Neumann propuso una mejora revolucionaria: almacenar los programas en la misma memoria que los datos. Este concepto se aplicó en la EDVAC y luego en muchas otras computadoras. En paralelo, Konrad Zuse desarrolló en Alemania la Z3, una máquina completamente automática y programable, aunque menos conocida.

La historia continuó con la invención del transistor (1947), que reemplazó a los tubos de vacío y permitió reducir el tamaño y consumo de las máquinas (ver Figura 4.5). Este cambio marcó el inicio de la llamada segunda generación de computadoras.

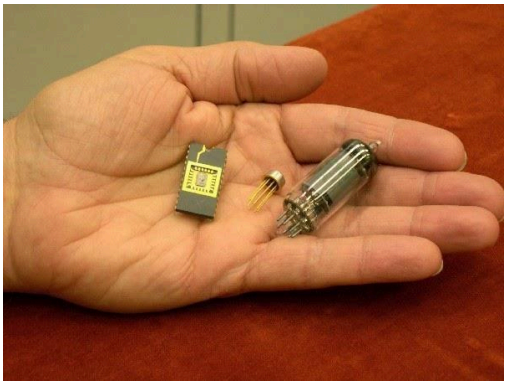

Figura 4.5: De derecha a izquierda: un tubo de vacío, un transistor y un chip.

Para organizar esta evolución, se suele hablar de generaciones:

- **Primera generación** (1940–1958): tubos de vacío, grandes dimensiones, uso de tarjetas perforadas.
- **Segunda generación** (1958–1964): transistores, menor consumo, mayor velocidad.
- **Tercera generación** (1964–1971): circuitos integrados, llegada del sistema operativo.
- **Cuarta generación** (1971–actualidad): microprocesadores, computadoras personales, interfaces gráficas.

Gracias a estos avances, la programación también evolucionó. Pasamos de operar máquinas con cables y tableros a escribir instrucciones comprensibles en lenguajes de programación. Este recorrido nos permite valorar el enorme desarrollo que ha tenido la computación y entender mejor cómo llegamos a las tecnologías actuales.

5. LA PROGRAMACIÓN.

Mencionamos anteriormente que la programación consistía en instruir a una computadora para que resuelva un problema y que la comunicación de esas instrucciones debe ser realizada de forma clara. Es por eso que, ante un problema que debe ser resuelto computacionalmente, el primer paso es pensar detalladamente cuál puede ser una forma de resolverlo, es decir, crear un algoritmo. Un algoritmo es una estrategia consistente de un conjunto ordenado de pasos que nos lleva a la solución de un problema o alcance de un objetivo. Luego, hay que traducir el algoritmo elegido al idioma de la computadora.

Entonces, podemos decir que la resolución computacional de un problema consiste de dos etapas básicas:

- **Diseño algorítmico:** desarrollar un algoritmo, o elegir uno existente, que resuelva el problema.
- **Codificación:** expresar un algoritmo en un lenguaje de programación para que la computadora lo pueda interpretar y ejecutar.

Al aprender sobre programación, comenzamos enfrentándonos a problemas simples para los cuales la primera etapa parece sencilla, mientras que la codificación se torna dificultosa ya que hay que aprender las reglas del lenguaje de programación (sintaxis del lenguaje). Sin embargo, mientras que con práctica rápidamente podemos ganar facilidad para la escritura de código, el diseño algorítmico se torna cada vez más desafiante al encarar problemas más complejos. Es por eso que haremos hincapié en el planteo y desarrollo de algoritmos como una etapa fundamental en la programación.

5.1. Diseño Algorítmico

Cotidianamente, hacemos uso de algoritmos para llevar adelante casi todas las actividades que realizamos: preparar el desayuno, sacar a pasear la mascota, poner en la tele un servicio de streaming para ver una película, etc. Cada una de estas tareas requiere llevar adelante algunas acciones de forma ordenada, aunque no hagamos un listado de las mismas y procedamos casi sin pensar.

Sin embargo, cuando estamos pensando la solución para un problema que va a resolver una computadora, debemos ser claros y concretos, para asegurarnos de que al seguir los pasos del algoritmo se llegue a la solución y para que quien tenga que codificarlo, nosotros mismos u otras personas, lo pueda entender sin problemas. Por eso, el primer paso es idear un algoritmo para su solución y expresarlo por escrito, por ejemplo, en español, pero adaptando el lenguaje humano a formas lógicas que se acerquen a las tareas que puede realizar una computadora.

En programación, el lenguaje artificial e informal que usan los desarrolladores en la confección de algoritmos recibe el nombre de pseudocódigo. Es la herramienta que utilizamos para describir los algoritmos mezclando el lenguaje común con instrucciones de programación. No es en sí mismo un lenguaje de programación, es decir, la computadora no es capaz de entenderlo, sino que el objetivo del mismo es que el programador se centre en la

solución lógica y luego lo utilice como guía al escribir el programa.

El pseudocódigo, como cualquier otro lenguaje, está compuesto por:

- Un léxico: conjunto de palabras o frases válidas para escribir las instrucciones.
- Una sintaxis: reglas que establecen cómo se pueden combinar las distintas partes.
- Una semántica: significado que se les da a las palabras o frases.

Por ejemplo, si nuestro problema es poner en marcha un auto, el algoritmo para resolverlo puede ser expresado en Pseudocódigo como se muestra en la siguiente Figura 5.1. Pero también los algoritmos suelen ser representados mediante diagramas de flujo como en la Figura 5.2.

```
Proceso ArrancarAuto
    Escribir "Insertar llave de contacto"
    Escribir "Ubicar el cambio en punto muerto"
    Escribir "Girar la llave hasta la posición de arranque"

    Escribir "¿El motor arranca? (SI/NO)"
    Leer respuesta

    Si respuesta = "SI" Entonces
        Escribir "Poner el cambio en primera"
    Sino
        Escribir "Llamar al mecánico"
    FinSi

    Escribir "Fin del proceso"
FinProceso
```

Figura 5.1: Ejemplo del algoritmo “Arrancar el auto” representado con Pseudocódigo.

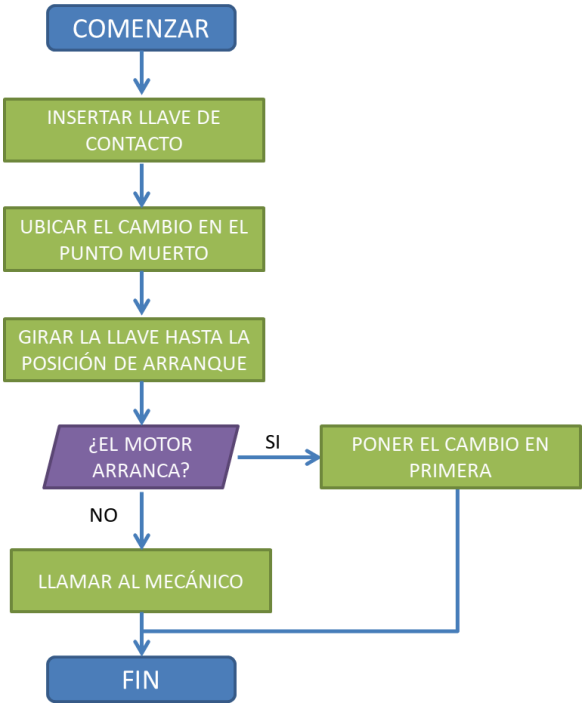

Figura 5.2: Ejemplo del algoritmo “Arrancar el auto” representado gráficamente con un diagrama de flujo.

5.2. Codificación

El algoritmo anterior, de la Figura 5.1, está presentado en pseudocódigo utilizando el lenguaje español, una opción razonable para compartir esta estrategia entre personas que se comuniquen con este idioma. Claramente, si queremos presentarle nuestro algoritmo a alguien que sólo habla francés, el español ya no sería una buena elección, y mucho menos si queremos presentarle el algoritmo a una computadora. Para que una computadora pueda entender nuestro algoritmo, debemos traducirlo en un lenguaje de programación, que, como dijimos antes, es un idioma artificial diseñado para expresar cómputos que puedan ser llevados a cabo por equipos electrónicos, es decir es un medio de comunicación entre el humano y la máquina.

Si bien hay distintos lenguajes de programación, una computadora en definitiva es un aparato que sólo sabe hablar en binario, es decir, sólo interpreta señales eléctricas con dos estados posibles, los cuales son representados por los dígitos binarios 0 y 1. Toda instrucción que recibe la computadora se construye mediante una adecuada y larga combinación de ceros y unos. Este sistema de código con ceros y unos que la computadora interpreta como instrucciones o conjuntos de datos se llama lenguaje de máquina (o código de máquina).

Programar en lenguaje de máquina es muy complejo y lento, y es fácil cometer errores pero es difícil arreglarlos. Por eso a principios de la década de 1950 se inventaron los lenguajes ensambladores, que usan palabras para representar simbólicamente las operaciones que debe realizar la computadora. Cada una de estas palabras reemplaza un código de máquina

binario, siendo un poco más fácil programar. Imaginemos que deseamos crear un programa que permita sumar dos números elegidos por una persona. La computadora puede hacer esto si se lo comunicamos mediante un mensaje compuesto por una larga cadena de ceros y unos (lenguaje de máquina) que a simple vista no podríamos entender. Sin embargo, escrito en lenguaje ensamblador, el programa se vería así (por ejemplo):

```
1  mov ah,01h  ;Se guarda 01h en el registro ah
2  int 21h     ;Se llama a la interrupción 21h
3  sub al,30h  ;Se resta 30h para obtener el número ingresado
4  mov var1,al ;Se guarda el número ingresado en var1
5  mov ah,01h  ;Se guarda 01h en el registro ah
6  int 21h     ;Se llama a la interrupción 21h
7  sub al,30h  ;Se resta 30h para obtener el número ingresado
8  add al,var1 ;Se suma var1 y al y el resultado se guarda en el registro al
9  mov dl,al   ;Se guarda el contenido del registro al en el registro dl
10 add dl,30h  ;Se suma 30h al registro dl para obtener la suma en ASCII
11 mov ah,02h  ;Se guarda 02h en el registro ah
12 int 21h     ;Se llama a la interrupción 21h
```

Figura 1.8: Programa en lenguaje ensamblador para leer dos números, sumarlos y mostrar el resultado. Al final de cada línea hay una descripción de la operación realizada.

El programa que se encarga de traducir esto al código de máquina se llama ensamblador. A pesar de que no haya ceros y unos como en el lenguaje de máquina, probablemente el código anterior tampoco sea fácil de entender. Aparecen instrucciones que tal vez podemos interpretar, como add por sumar o sub por substraer, pero está lleno de cálculos hexadecimales, referencias a posiciones en la memoria de la computadora y movimientos de valores que no lo hacen muy amigable. Por eso, a pesar de que la existencia de los lenguajes ensambladores simplificó mucho la comunicación con la computadora, se hizo necesario desarrollar lenguajes que sean aún más sencillos de usar.

Por ejemplo, con el lenguaje que vamos a aprender, Phyton, el problema de pedirle dos números a una persona y sumarlos se resumen en las siguientes líneas de código:

```
num1 = int(input("Ingrese número: "))
num2 = int(input("Ingrese número: "))
print(num1 + num2)
```

En las dos primeras líneas con la instrucción input() (que quiere decir “ingresar”, “leer”) se le pide a la persona que indique dos números y en la tercera línea se muestra el resultado de la suma, con la instrucción print() (“imprimir”, “mostrar”). Mucho más corto y entendible. En este caso sumamos números enteros (int), pero puedo cambiar para sumar reales (float).

Esta simplificación es posible porque nos permite ignorar ciertos aspectos del proceso que realiza la computadora. Todas esas acciones que se ven ejemplificadas en la imagen con el código ensamblador se llevan a cabo de todas formas, pero no lo vemos. Nosotros sólo tenemos que aprender esas últimas tres líneas de código, de forma que nos podemos concentrar en el problema a resolver (ingresar dos números, sumarlos y mostrar el resultado) y no en las complejas operaciones internas que tiene que hacer el microprocesador.

En programación, la idea de simplificar un proceso complejo ignorando algunas de sus partes para comprender mejor lo que hay que realizar y así resolver un problema se conoce como abstracción. Esto quiere decir que los lenguajes de programación pueden tener distintos niveles de abstracción:

- **Lenguajes de bajo nivel de abstracción**: permiten controlar directamente el hardware de la computadora, son específicos para cada tipo de máquina, y son más rígidos y complicados de entender para nosotros. El lenguaje ensamblador entra en esta categoría.
- **Lenguajes de alto nivel de abstracción**: diseñados para que sea fácil para los humanos expresar los algoritmos sin necesidad de entender en detalle cómo hace exactamente el hardware para ejecutarlos. Son independientes del tipo de máquina.
- **Lenguajes de nivel medio de abstracción**: son lenguajes con características mixtas entre ambos grupos anteriores.

Figura 1.9: Distintos lenguajes de programación y sus logos.

Si bien podemos programar usando un lenguaje de alto nivel para que nos resulte más sencillo, alguien o algo debe traducirlo a lenguaje de máquina para que la computadora, que sólo entiende de ceros y unos, pueda realizar las tareas. Esto también es necesario incluso si programamos en lenguaje ensamblador. Para estos procesos de traducción se crearon los compiladores e intérpretes.

Un compilador es un programa que toma el código escrito en un lenguaje de alto nivel y lo traduce a código de máquina, guardándolo en un archivo que la computadora ejecutará posteriormente (archivo ejecutable). Para ilustrar el rol del compilador, imaginemos que alguien que sólo habla español le quiere mandar una carta escrita en español a alguien que vive en Alemania y sólo habla alemán. Cuando esta persona la reciba, no la va a entender. Se necesita de un intermediario que tome la carta en español, la traduzca y la escriba en alemán y luego se la mande al destinatario, quien ahora sí la podrá entender. Ese es el rol de un compilador en la computadora. Ahora bien, el resultado de la traducción, que es la carta escrita en alemán, sólo sirve para gente que hable alemán. Si se quiere enviar el mismo mensaje a personas que hablen otros idiomas, necesitaremos hacer la traducción que

corresponda. De la misma forma, el código generado por un compilador es específico para cada máquina, depende de su arquitectura.

Además de los compiladores, para realizar este pasaje también existen los intérpretes. Un intérprete es un programa que traduce el código escrito en lenguaje de alto nivel a código de máquina, pero lo va haciendo a medida que se necesita, es decir, su resultado reside en la memoria temporal de la computadora y no se genera ningún archivo ejecutable. Siguiendo con el ejemplo anterior, es similar a viajar a Alemania con un intérprete que nos vaya traduciendo en vivo y en directo cada vez que le queramos decir algo a alguien de ese país. Python es interpretado porque no necesita compilarse completamente antes de ejecutarse. El intérprete hace ese trabajo internamente.

Concluyendo, gracias al concepto de la abstracción podemos escribir programas en un lenguaje que nos resulte fácil entender, y gracias al trabajo de los compiladores e intérpretes la computadora podrá llevar adelante las tareas necesarias.

Cada una de las acciones que componen al algoritmo son codificadas con una o varias instrucciones, expresadas en el lenguaje de programación elegido, y el conjunto de todas ellas constituye un programa. El programa se guarda en un archivo con un nombre generalmente dividido en dos partes por un punto, por ejemplo: **miPrimerPrograma.py**. La primera parte es la raíz del nombre con la cual podemos describir el contenido del archivo. La segunda parte es indicativa del uso del archivo, por ejemplo, indica que contiene un programa escrito en lenguaje Phyton. El proceso general de ingresar o modificar el contenido de un archivo se denomina edición.

5.3. Errores de programación

Apenas iniciamos nuestro camino en el mundo de la programación nos daremos cuenta que tendremos siempre ciertos compañeros de viaje: los errores. Muchas veces nos pasará que queremos ejecutar nuestro código y el mismo no anda o no produce el resultado esperado. No importa cuán cuidadosos seamos, ni cuánta experiencia tengamos, los errores están siempre presentes. Con el tiempo y práctica, vamos a poder identificarlos y corregirlos con mayor facilidad, pero probablemente nunca dejemos de cometerlos.

A los errores en programación se los suele llamar bugs (insecto o bicho en inglés) y el proceso de la corrección de los mismos se conoce como debugging (depuración). Se dice que esta terminología proviene de 1947, cuando una computadora en la Universidad de Harvard (la Mark II) dejó de funcionar y finalmente se descubrió que la causa del problema era la presencia de una polilla en un relé electromagnético de la máquina. Sin embargo, otros historiadores sostienen que el término ya se usaba desde antes.

A continuación, se presenta una de las posibles clasificaciones de los errores que se pueden cometer en programación:

1. **Errores de sintaxis.** Tal como el lenguaje humano, los lenguajes de programación tienen su propio vocabulario y su propia sintaxis, que es el conjunto de reglas gramaticales que establecen cómo se pueden combinar las distintas partes. Estas reglas sintácticas determinan que ciertas instrucciones están correctamente construidas, mientras que otras no. Cuando ejecutamos un programa, el compilador o el intérprete chequea si el mismo es sintácticamente correcto. Si

hemos violado alguna regla, por ejemplo, nos faltó una coma o nos sobra un paréntesis, mostrará un mensaje de error y debemos editar nuestro programa para corregirlo. En estos casos, hay que interpretar el mensaje de error, revisar el código y corregir el error.

2. **Errores lógicos.** Se presentan cuando el programa puede ser compilado sin errores pero arroja resultados incorrectos o ningún resultado. El software no muestra mensajes de error, debido a que, por supuesto, no sabe cuál es el resultado deseado, sino que sólo se limita a hacer lo que hemos programado. En estos casos hay que revisar el programa para encontrar algún error en su lógica. Este tipo de errores suelen ser los más problemáticos. Algunas ideas para enfrentarlos incluyen volver a pensar paso por paso lo que se debería hacer para solucionar el problema y compararlo con lo que se ha programado, agregar pasos para mostrar resultados intermedios o emplear herramientas especializadas de debugging (llamadas debugger) para explorar el código paso a paso hasta identificar el error.
3. **Errores en la ejecución (runtime errors).** Se presentan cuando el programa está bien escrito, sin errores lógicos ni sintácticos, pero igualmente se comporta de alguna forma incorrecta. Se dan a pesar de que el programa ande bien en el entorno de desarrollo del programador, pero no cuando algún usuario lo utiliza en algún contexto particular. Puede ser que se intente abrir un archivo que no existe, que el proceso supere la memoria disponible, que tomen lugar operaciones aritméticas no definidas como la división por cero, etc.

Los errores en la programación son tan comunes, que un científico de la computación muy reconocido, Edsger Dijkstra, dijo una vez: “si la depuración es el proceso de eliminar errores, entonces la programación es el proceso de generarlos”. Ante la presencia de uno, no hay más que respirar profundo y con paciencia revisar hasta encontrarlo y solucionarlo.