REPÚBLICA DEL ECUADOR

Ministerio de Educación

UNIDAD EDUCATIVA "BAÑOS"

Baños de Agua Santa – Tungurahua

PROYECTO INTERMODULAR

"Temas de Primero de Bachillerato Currículo del Módulo Formativo Nº3: Programación y Base de Datos"

Figura Profesional: Informática

Elaborado por: Anahi Sailema

Pamela Aponte

Elkyn Zambrano

Curso: Segundo "H"

Módulo formativo:

Módulo 3: Programación y Base de Datos

Docentes Técnicos: Mg. Diana Fuentes

Año Lectivo: 2024-2025

EL NUEVO ECUADOR

Índice

Índice 2
1. Programación 3
1.1 Datos 3
1.3 Pseudocódigo 6
1.4 Flujograma 8
1.5 Estructuras de control 10
1.6 Operadores 11
1.7 Funciones 13
1.8 Estructura modular de programas 15
2. Lenguaje de programación 19
2.1 Entidades que maneja el lenguaje: 19
2.2 Tipos de variables 20
2.3 Estructuras de datos 21
2.4 Instrucciones del lenguaje 22
2.5 Función y sintaxis 24
2.6 Funciones y librerías básicas 25
2.7 Documentación de programas 27
2.8 Descripción de estructuras de datos utilizadas 29
2.9 Código fuente 31

1. Programación

La programación es el **arte de organizar un conjunto de instrucciones que las computadoras siguen para realizar tareas específicas.** Es fundamental para la comunicación entre usuarios y máquinas, y es la base de todas las aplicaciones y programas que usamos a diario. (Executrain, 2022)

1.1 Datos: En programación, los datos son valores o información que se utilizan para realizar cálculos, tomar decisiones o almacenar información. Los datos pueden ser de diferentes tipos, como números enteros, números decimales, texto, fechas, horas, etc.

Por lo tanto, los valores o referentes que recibe un sistema informático a través de distintos medios, y que una vez interpretados se convierten en información. Para ello, son procesados por diferentes algoritmos de programación.

En los lenguajes de programación, empleados para crear y organizar los algoritmos que todo sistema informático o computacional persigue, los datos son la expresión de las características puntuales de las entidades sobre las cuales operan dichos algoritmos. Es decir, son el input inicial, a partir del cual puede procesarse y componerse la información.

Tipos de datos comunes:

1. Números enteros (int): 1, 2, 3, etc.

2. Números decimales (float): 3.14, -0.5, etc.

3. Texto (string): "hola", "adiós", etc.

4. Booleanos (bool): verdadero (true), falso (false)

5. Fechas y horas (date, datetime): 2022-07-25, 14:30:00, etc.

Ejemplo:

Supongamos que queremos crear un programa que calcule el área de un rectángulo. Para hacer esto, necesitamos almacenar los datos de la longitud y el ancho del rectángulo.

```
// Declaración de variables
longitud = 5; // número entero
ancho = 3.5; // número decimal
// Cálculo del área
área = longitud * ancho;
// Impresión del resultado
print("El área del rectángulo es:", área);
```

En este ejemplo, los datos son:

- longitud: un número entero (5)
- ancho: un número decimal (3.5)
- área: el resultado del cálculo, que también es un número decimal (Concepto, 2025)

1.2 Algoritmo: Un algoritmo es un conjunto de instrucciones paso a paso que se utilizan para resolver un problema o realizar una tarea específica en programación. Un algoritmo define la lógica y la secuencia de operaciones que se deben realizar para obtener un resultado deseado.

En **el campo de la programación informática**, los algoritmos son conjuntos de reglas que indican al ordenador cómo ejecutar una tarea. En realidad, un programa informático es un algoritmo que indica al ordenador qué pasos debe realizar y en qué orden para llevar a cabo una tarea específica. Se escriben utilizando un lenguaje de programación.

Características de un algoritmo:

1. Secuencia de pasos: Un algoritmo se compone de una serie de pasos que se deben seguir en un orden específico.

2. Entradas y salidas: Un algoritmo recibe entradas (datos) y produce salidas (resultados).

3. Lógica y toma de decisiones: Un algoritmo utiliza lógica y toma de decisiones para procesar las entradas y producir las salidas.

¿Cómo se utilizan los algoritmos en informática?

En informática, los algoritmos son omnipresentes. De hecho, son la columna vertebral de la informática, ya que un algoritmo da al ordenador un conjunto específico de instrucciones.

Esas instrucciones son las que permiten que el ordenador realice las tareas. Los programas informáticos son, a su vez, algoritmos escritos en lenguajes de programación.

Los algoritmos también desempeñan un papel fundamental en el funcionamiento de las redes sociales, por ejemplo. Deciden qué publicaciones se muestran o qué anuncios se ofrecen al usuario.

Ejemplo:

Supongamos que queremos crear un algoritmo para calcular el mayor de dos números.

// Algoritmo para calcular el mayor de dos números

1. Recibir dos números como entradas (a y b)

2. Comparar a y b

3. Si a es mayor que b, entonces:

- Salida: a es el mayor

4. Si b es mayor que a, entonces:

- Salida: b es el mayor

5. Si a y b son iguales, entonces:

- Salida: a y b son iguales

En este ejemplo, el algoritmo:

- Recibe dos números como entradas (a y b)
- Compara a y b utilizando lógica y toma de decisiones
- Produce una salida que indica cuál es el mayor de los dos números (DataScientest, 2025)

1.3 Pseudocódigo: Un pseudocódigo es una representación simbólica y simplificada de un algoritmo o un programa de computadora. Se utiliza para describir la lógica y la secuencia de operaciones de un programa de manera clara y concisa, sin utilizar un lenguaje de programación específico.

Diferencia con el código real

Una de las diferencias clave entre el pseudocódigo y el código real es que el pseudocódigo no sigue una sintaxis formal. Mientras que los lenguajes de programación como Python, Java o C++ requieren un formato específico y reglas sintácticas, el pseudocódigo es mucho más flexible y se escribe en un estilo más parecido al lenguaje humano, lo que facilita su comprensión. Aunque imita la estructura del código, no tiene

las restricciones del lenguaje de programación, lo que lo hace útil como herramienta de planificación.

El pseudocódigo también es más abstracto que el código real, ya que no necesita incluir detalles específicos como la gestión de memoria o la definición de tipos de datos, lo que permite centrarse exclusivamente en la lógica del programa.

Propósito del pseudocódigo

El principal objetivo del pseudocódigo es ayudar a los desarrolladores a planificar y estructurar la lógica de un programa antes de escribir el código real. Al separar la lógica de la sintaxis del lenguaje de programación, el pseudocódigo permite a los programadores concentrarse en el diseño del algoritmo sin preocuparse por los detalles técnicos, lo que reduce el riesgo de cometer errores lógicos más adelante.

Además, es una herramienta excelente para la comunicación entre programadores, ya que facilita que todos los miembros del equipo comprendan la lógica de un sistema antes de implementarlo. También es útil para los estudiantes de programación, ya que les permite practicar la resolución de problemas sin las complicaciones adicionales de un lenguaje específico.

Características de los pseudocódigos:

1. Simbólico: Los pseudocódigos utilizan símbolos y abreviaturas para representar las operaciones y los datos.
2. Simplificado: Los pseudocódigos omiten detalles innecesarios y se enfocan en la lógica y la secuencia de operaciones.
3. Independiente del lenguaje: Los pseudocódigos no están vinculados a un lenguaje de programación específico.
4. Fácil de entender: Los pseudocódigos están diseñados para ser fáciles de entender y analizar.

Ejemplo sencillo de pseudocódigo

- Ejemplo de pseudocódigo básico:

```
INICIO
   LEER número1
   LEER número2
   SI número1 > número2 ENTONCES
      IMPRIMIR "El número mayor es número1"
   SINO
      IMPRIMIR "El número mayor es número2"
   FIN_SI
FIN
```

Este ejemplo muestra cómo el pseudocódigo sigue una estructura simple y clara, describiendo los pasos lógicos de manera comprensible sin depender de un lenguaje de programación específico. (Robledano, 2019)

1.4 Flujograma: Un flujograma o diagrama de flujo es una herramienta gráfica que representa visualmente un proceso o un algoritmo. Es muy utilizado en los campos de la informática, la economía, la industria e incluso la psicología, para organizar de un modo simple las decisiones y los pasos involucrados en algún tipo de proceso.

Elementos del diagrama de flujo

Principalmente, un diagrama de flujos está formado por diferentes elementos que nos permiten dar forma a la idea del algoritmo:

- **Línea o flechas del flujo:** Indica la instrucción que se va a realizar, o la dirección del flujo del proceso.

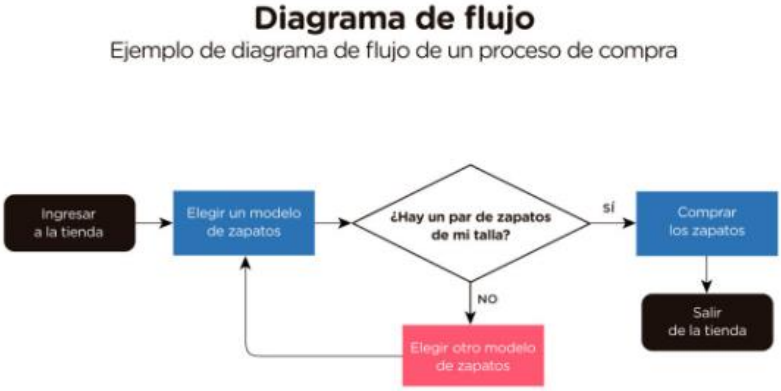
- **Inicio y final:** Es la forma en la cual se indica el “inicio del proceso” y “final del proceso”. Su icono suele ser un rectángulo con las esquinas redondeadas.
- **Asignación/ definición:** Permite asignar un valor o definir una variable, son los procesos o actividades que se necesitan para resolver el problema. En este caso, suele estar representado con un rectángulo.
- **Datos de entrada o de salida:** Representa la impresión de datos en la salida. Se representa con un recuadro con las esquinas inclinadas y una flecha hacia fuera.
- **Decisión:** Indica que des del punto que nos encontramos, puede haber más de un camino para seguir, según la condición dada. En este caso se usa un rombo.

Ejemplo:

Ejemplo de flujograma de un proceso de compra:

Figura 1

Flujograma



Nota: Se puede observar un flujograma de un proceso de compras. Fuente: (Concepto,2025).

(Epitech, 2021)

1.5 Estructuras de control: Las estructuras de control son un conjunto de reglas que hacen posible la gestión del flujo de ejecución de los programas, esto lo logran determinando el orden en que las instrucciones se ejecutan. *Grosso modo*, las estructuras de control permiten que el código tome decisiones, repita acciones o bien siga su flujo de ejecución secuencial.

Características de los frameworks

Algunos de los elementos y características comunes que incluyen son:

- **Enrutamiento:** suelen proporcionar un sistema de enrutamiento que facilita la gestión de las URL y las rutas de las páginas dentro de una aplicación.
- **Plantillas:** permiten crear un sitio web de manera más sencilla mediante el uso de plantillas predefinidas o motores de plantillas que ayudan a generar el código HTML de manera dinámica.
- **Controladores:** a menudo utilizan el patrón de diseño MVC (modelo-vista-controlador) o una variante de él, lo que significa que facilitan la separación de la lógica de negocio (controlador), de la presentación (vista) y los datos (modelo).
- **Manejo de solicitudes y respuestas:** ofrecen funciones para gestionar solicitudes HTTP entrantes y generar respuestas HTTP, lo que simplifica la interacción con el cliente.
- **Bases de datos:** muchos entornos integran herramientas y abstracciones para interactuar con bases de datos, lo que facilita el almacenamiento y recuperación de los mismos.
- **Seguridad:** suelen incluir mecanismos de seguridad incorporados para proteger contra amenazas tales como los ataques de inyección SQL o ataques de scripting entre sitios (XSS).

- **Sesiones y autenticación:** proporcionan funciones para gestionar las sesiones de usuario y su autenticación, lo que facilita la creación de sistemas de inicio de sesión y control de acceso.
- **Middleware:** permiten la inclusión de componentes de esta clase que pueden realizar tareas adicionales como la compresión de respuestas, el registro de solicitudes o la autenticación en el flujo de la aplicación.
- **API y servicios web:** facilitan la creación y consumo de estas interfaces y servicios, lo que permite la integración con otras aplicaciones y servicios.

Ejemplo de estructura de control secuencial:

Las estructuras de control secuenciales son las más básicas. En este caso, las instrucciones se ejecutan una tras otra en el mismo orden en que están escritas. Piensa en este tipo de estructuras como una fila de acciones que siguen un patrón fijo. Estas son la base de cualquier algoritmo, en donde cada acción sigue a la anterior de modo predecible. Veamos un ejemplo en JavaScript:

```
// Ejemplo en JavaScript
let nombre = "María";
let saludo = "Hola, " + nombre + "!";
console.log(saludo);
```

En este ejemplo, la primera línea asigna un valor a la variable nombre, luego se crea un saludo y finalmente se imprime en la consola. Todas las acciones ocurren en el orden en que fueron escritas. (Frisoli, 2023)

1.6 Operadores: En este ejemplo, la primera línea asigna un valor a la variable nombre, luego se crea un saludo y finalmente se imprime en la consola. Todas las acciones ocurren en el orden en que fueron escritas.

Los operadores se utilizan para construir expresiones, que son combinaciones de operandos y operadores que producen un resultado. Por ejemplo, en la expresión a + b, el operador + indica una suma entre los operandos a y b. Existen varios tipos de operadores en programación, incluyendo:

1. Operadores aritméticos: como suma (+), resta (-), multiplicación (*), división (/), etc.
2. Operadores de comparación: como igualdad (==), desigualdad (!=), mayor que (>), menor que (<), etc.
3. Operadores de asignación: como asignación (=), asignación con suma (+=), asignación con resta (-=), etc.
4. Operadores lógicos: como AND (&&), OR (||), NOT (!), etc.
5. Operadores de incremento y decremento: como incremento (++), decremento (--), etc.
6. Operadores de concatenación: como el operador de concatenación de cadenas en algunos lenguajes (.), etc.

Ejemplo:

Operadores aritméticos que se utilizan para realizar operaciones matemáticas en programación.

- **+ (suma):** Se utiliza para sumar dos valores.
- **- (resta):** Se utiliza para restar el valor de la derecha del valor de la izquierda.
- ***** **(multiplicación):** Se utiliza para multiplicar dos valores.
- **/ (división):** Se utiliza para dividir el valor de la izquierda por el valor de la derecha.
- **% (módulo):** Devuelve el resto de la división del valor de la izquierda por el valor de la derecha.

- **// (división entera):** Devuelve el cociente entero de la división del valor de la izquierda por el valor de la derecha.
- **** (exponenciación):** Se utiliza para elevar el valor de la izquierda a la potencia del valor de la derecha.

Figura 2

Operadores aritméticos

Nota: Ejemplo operador aritmético «suma +». Fuente: (Apinem,2025).

(Aderformacion, 2025)

1.7 Funciones: En programación, una función es un conjunto de instrucciones que realiza una tarea específica. Puedes imaginar una función como una pequeña máquina que toma ciertos datos (argumentos), realiza una operación y devuelve un resultado. Utilizar funciones nos permite escribir código más limpio, organizado y fácil de entender.

Principales características de las funciones en la programación.

Además de las características claves nombradas en el punto 1 como la sintaxis, nombre, parámetros, cuerpo, reutilización, etc. Algunos otros puntos claves de las funciones son:

- Ámbito (Scope):

  El ámbito de una variable define dónde puede ser accedida (ámbito local o global). Las variables definidas dentro de una función tienen un ámbito local, mientras que las definidas fuera de la función tienen un ámbito global.

- Reutilización de código:

Una de las características más importantes de las funciones es su capacidad para promover la reutilización de código. Puedes definir una tarea específica en una función y llamarla en diferentes partes del programa.

- Modularidad:

  Las funciones permiten organizar el código en módulos más pequeños y manejables. Cada función se centra en una tarea específica, facilitando el mantenimiento y la comprensión del código.

- Recursividad:

  Una función puede llamarse a sí misma, lo que se conoce como recursividad. Esto es útil para implementar algoritmos que se dividen en subproblemas más pequeños.

- Manejo de errores:

  Pueden incluir bloques de manejo de errores, proporcionando una forma estructurada de lidiar con situaciones excepcionales.

- Funciones anidadas y Cierres:

  Algunos lenguajes permiten funciones anidadas, que son funciones dentro de funciones. Además, las funciones pueden formar cierres, lo que significa que pueden recordar el ámbito en el que fueron creadas.

- Decoradores:

  Algunos lenguajes de programación permiten el uso de decoradores, que son funciones especiales que modifican o extienden el comportamiento de otras funciones.

Ejemplo:

```
# Ejemplo de una función en Python
def saludar(nombre):
    return "¡Hola, " + nombre + "!"
# Llamada a la función
```

```
mensaje = saludar("Ichibytes")
print(mensaje)
```

En este ejemplo, la función saludar toma un argumento (nombre) y devuelve un saludo personalizado. (Epinem, 2025)

1.8 Estructura modular de programas: La estructura modular de programas es una técnica de diseño de software que consiste en dividir un programa en módulos o unidades independientes, cada uno con una función específica. Cada módulo es un conjunto de instrucciones que realizan una tarea particular y pueden ser utilizados por otros módulos o por el programa principal.

La estructura de un programa modular consta de un módulo principal desde el que se llamará al resto de los módulos. El módulo principal recibe el control al inicio de la ejecución del programa. Cuando se invoca un módulo concreto (a través de su nombre y parámetros), el control del programa se pasará al módulo. Este módulo mantendrá el control hasta que no se finalice su ejecución en cuyo momento devolverá el control a la instrucción siguiente a la que realizó la llamada.

Para realizar una programación modular podemos emplear algunas técnicas, entre ellas las más utilizadas: Programación Descendente y Programación Ascendente.

Programación Descendente

En la programación descendente primero se define el módulo principal y después las restantes llamadas a módulos específicos. Cuando se lleva a cabo un diseño descendente es lógico pensar en algún tipo de jerarquía. Cuando esto ocurre es frecuente que la planeación resultante culmine con un diagrama de diseño estructurado. Este último es diferente del diagrama de flujo convencional ya que el diagrama de diseño estructurado es más general y señala el sentido de orden, responsabilidad y control que existe en el código de un programa. En este diagrama

cada módulo está representado por un rectángulo que contiene un número, el cual indica la posición del módulo dentro del esquema jerárquico del programa. El único módulo que puede existir con un nivel de jerarquía cero es el módulo principal del programa. Después puede tenerse hasta tres módulos de nivel-1. Estos módulos reflejan las tareas de entrada, salida y procesamiento de información llevadas a cabo por el programa.

Programación Ascendente

La técnica BOTTOM UP es conocida también como ascendente, la diferencia entre el bottom up y el top down es que los módulos son enumerados de forma diferente. En el bottom up se enumeran primero los módulos inferiores hasta llegar al módulo superior.

La diferencia del tipo de diseño ascendente y descendente solo se puede dar a la hora de la programación. Porque en el momento de dibujar la estructura del problema, en las dos formas el diseño queda igual, solamente que los módulos son enumerados en forma diferente, pero esto se hace pensando ya en cómo se va a comenzar a programar. En el diseño ascendente primero se programan los módulos que se encuentran más abajo de la estructura, hasta llegar al primer módulo dibujado.

Datos locales y globales

Datos locales que son ocultos en el interior de un módulo y son utilizados, exclusivamente, por el módulo. Estos datos locales están estrechamente relacionados con sus módulos y están protegidos de modificaciones por otras funciones.

Otro tipo de datos son los datos globales a los cuales se puede acceder desde cualquier módulo del programa. Es decir, dos o más módulos pueden acceder a los

mismos datos siempre que estos datos sean globales. En la Figura 1 se muestra la disposición de variables locales y globales en un programa procedimental.

Características de la estructura modular

1. Independencia: Cada módulo es independiente y puede ser modificado o reemplazado sin afectar a otros módulos.

2. Reutilización: Los módulos pueden ser reutilizados en otros programas o proyectos.

3. Flexibilidad: La estructura modular permite agregar o eliminar módulos según sea necesario.

4. Mantenimiento: La estructura modular facilita el mantenimiento y la depuración de los programas.

Ejemplo de estructura modular

Supongamos que queremos crear un programa que realice las siguientes tareas:

1. Leer un archivo de texto

2. Procesar el contenido del archivo

3. Guardar el resultado en un nuevo archivo

Podemos dividir el programa en tres módulos independientes:

Módulo 1: Leer archivo

```
def leer_archivo(nombre_archivo):
# Código para leer el archivo
 return contenido
```

Módulo 2: Procesar contenido

```
def procesar_contenido(contenido):
# Código para procesar el contenido
return resultado
```

Módulo 3: Guardar resultado

```
def guardar_resultado(resultado, nombre_archivo):

# Código para guardar el resultado en un nuevo archivo

Programa principal

def main():

nombre_archivo = "archivo.txt"

contenido = leer_archivo(nombre_archivo)

resultado = procesar_contenido(contenido)

guardar_resultado(resultado, "resultado.txt")

main()
```

En este ejemplo, cada módulo es independiente y realiza una tarea específica. El programa principal utiliza los módulos para realizar la tarea completa. (Guanajuato, 2023)

2. Lenguaje de programación

Se conoce como lenguaje de programación a un programa destinado a la construcción de otros programas informáticos. Su nombre se debe a que comprende un lenguaje formal que está diseñado para organizar algoritmos y procesos lógicos que serán luego llevados a cabo por un ordenador o sistema informático, permitiendo controlar así su comportamiento físico, lógico y su comunicación con el usuario humano. (Concepto, 2025)

2.1 Entidades que maneja el lenguaje: La mayoría de lenguajes de programación, aun cuando tengan diferente sintaxis y funcionalidades varias, tienen en común una serie de entidades que hacen posible su trabajo.

Estas entidades permiten indicar qué tiene que hacer un programa. Haciendo un símil podemos verlas como órdenes: estas entidades son las órdenes que disponemos para indicarle al ordenador lo que tiene que hacer. Aquí nos encontraremos variables, sentencias de control, bucles, funciones, etc.

Depende de cada lenguaje de programación, pero una variable es lo mismo en, prácticamente cualquier lenguaje. Variará tal vez su uso, cómo almacena un valor, etc., pero el concepto no variará.

Las entidades que maneja un lenguaje de programación son los elementos básicos que se utilizan para crear programas. Estas entidades pueden ser:

- Variables: Espacios de memoria que almacenan valores.
- Constantes: Valores que no cambian durante la ejecución del programa.
- Tipos de datos: Categorías que definen el tipo de valor que puede almacenar una variable.
- Operadores: Símbolos que se utilizan para realizar operaciones aritméticas, lógicas, etc.

- Estructuras de control: Instrucciones que controlan el flujo de ejecución del programa. (Aprendeitonline, 2025)

2.2 Tipos de variables: Los tipos de variables son las categorías que definen el tipo de valor que puede almacenar una variable. Algunos ejemplos de tipos de variables son:

- Enteros (int): Números enteros, como 1, 2, 3, etc.
- Números decimales (float): Números decimales, como 3.14, -0.5, etc.
- Cadenas de texto (string): Secuencias de caracteres, como "hola", "adiós", etc.
- Booleanos (bool): Valores lógicos, como verdadero (true) o falso (false).

¿Qué es una variable dentro de la programación?

Una variable es la asignación de un tipo de dato a una palabra con el objetivo de almacenarla dentro de la memoria, además de gestionar la información que decidas incorporar en el código. Estas palabras claves se pueden expresar mediante números, textos, objetos o, incluso, datos abstractos.

También hay que mencionar que los nombres de las variables también son conocidos como identificadores, ya que indican el tipo e incluso cuánto espacio de memoria tomará dicha variable.

Los principales usos para las variables son:

- **Guardar** datos y estados.
- **Asignar** valores de una variable a otra.
- **Representar** valores dentro de una expresión matemática.
- **Mostrar** valores por pantalla.

Ejemplo:

```
int edad = 25; // variable entera
float altura = 1.75; // variable decimal
string nombre = "Juan"; // variable cadena de texto
bool isAdmin = true; // variable booleana
```

(Blogticjob, 2023)

2.3 Estructuras de datos: Las estructuras de datos son formas de organizar y almacenar datos en un programa. Algunos ejemplos de estructuras de datos son:

- Arreglos (arrays): Colecciones de valores del mismo tipo que se almacenan en una secuencia.
- Listas (lists): Colecciones de valores del mismo tipo que se almacenan en una secuencia y pueden ser modificadas.
- Diccionarios (dictionaries): Colecciones de pares clave-valor que se almacenan en una tabla.

¿Para qué sirven las estructuras de datos?

En el ámbito de la informática, las **estructuras de datos** son aquellas que nos permiten, como desarrolladores, organizar la información de manera eficiente, y en definitiva diseñar la solución correcta para un determinado problema.

Ya sean las más utilizadas comúnmente como las variables, arrays, conjuntos o clases o las diseñadas para un propósito específico árboles, grafos, tablas, etc. una estructura de datos nos permite trabajar en un algo nivel de abstracción almacenando información para luego acceder a ella, modificarla y manipularla.

¿Cuáles son los tipos de estructuras de datos?

Primero, debemos diferenciar entre estructura de dato estática y estructura de dato dinámica

Las estructuras de datos estáticas son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa, mientras que una estructura de datos

dinámica es aquella en la que el tamaño ocupado en memoria puede modificarse durante la ejecución del programa.

Ejemplo:

```
int[] edades = {25, 30, 35}; // arreglo de enteros
string[] nombres = {"Juan", "María", "Pedro"}; // arreglo de cadenas de texto
Dictionary<string, int> edadPorNombre = new Dictionary<string, int>()
{
   {"Juan", 25},
   {"María", 30},
   {"Pedro", 35}
}; // diccionario de cadenas de texto y enteros
```

(Henry, 2022)

2.4 Instrucciones del lenguaje: Las instrucciones del lenguaje son las órdenes que se utilizan para controlar el flujo de ejecución del programa. Algunos ejemplos de instrucciones del lenguaje son:

- Instrucciones de asignación: Asignan un valor a una variable.

- Instrucciones de control de flujo: Controlan el orden en que se ejecutan las instrucciones.

- Instrucciones de iteración: Repiten un conjunto de instrucciones varias veces.

Cómo funcionan los lenguajes de programación

Instrucciones y algoritmos

En un lenguaje de programación, los desarrolladores crean programas al definir una serie de instrucciones que la computadora debe seguir. Estas instrucciones se organizan en algoritmos, que son conjuntos de pasos lógicos para resolver un problema.

Un algoritmo define cómo un problema será abordado por la máquina, desde la entrada de datos hasta la salida esperada. Los lenguajes de programación permiten estructurar estos algoritmos de forma clara y comprensible, facilitando la resolución de problemas complejos.

Compiladores e intérpretes

Los lenguajes compilados e interpretados difieren en la forma en que se ejecuta el código. Un compilador toma el código fuente completo y lo convierte en código máquina antes de ejecutarlo, como sucede en lenguajes como C o C++.

En cambio, un intérprete lee y ejecuta el código línea por línea, como ocurre en lenguajes como Python o JavaScript.

Tipos de lenguajes de programación

- **Lenguajes de bajo nivel**: Estos lenguajes, como el **ensamblador**, están más cercanos al **lenguaje de máquina** y permiten a los desarrolladores tener un control detallado sobre el hardware. Aunque ofrecen un rendimiento muy eficiente, son más difíciles de aprender y utilizar debido a su complejidad.
- **Lenguajes de alto nivel**: Este tipo de lenguajes de programación, como **Python**, **Java** y **C++**, son más cercanos al lenguaje humano, lo que facilita su aprendizaje y uso. Abstraen muchos detalles técnicos del hardware, permitiendo a los desarrolladores centrarse en resolver problemas sin preocuparse por la gestión directa de los recursos del sistema.
- **Lenguajes orientados a objetos**: Este otro tipo, como son **Java**, **C++** y **Python**, se basan en la creación de **clases** y **objetos**. Permiten estructurar el código de manera modular y reutilizable, lo que facilita el desarrollo y el mantenimiento de aplicaciones grandes. La POO introduce conceptos como **herencia**, **polimorfismo** y **encapsulación**, que permiten diseñar programas de forma más organizada.
- **Lenguajes funcionales**: Este cuarto tipo de lenguajes de programación, como **Haskell** o **Lisp**, están basados en **funciones matemáticas** y evitan el uso de

estados y efectos colaterales. Se enfocan en la creación de código más predecible y fácilmente depurable, ya que se evitan los cambios de estado en el flujo de ejecución. (Cimas Cuadrado, 2020)

2.5 **Función y sintaxis:** Una función es un bloque de código que se puede llamar varias veces desde diferentes partes del programa. La sintaxis de una función varía según el lenguaje de programación.

Elementos de la sintaxis

Existen varios **elementos** importantes relativos a la sintaxis en un lenguaje de programación y son los siguientes:

- **Palabras reservadas:** Son palabras que tienen un significado especial en el lenguaje de programación y se utilizan para realizar acciones específicas. Por ejemplo, "if", "else", "for", "while", "class", entre otras.
- **Identificadores:** Son nombres que se utilizan para identificar variables, funciones, clases, módulos u otros objetos en el código. Los identificadores deben seguir ciertas reglas como no empezar con números o incluir espacios.
- **Operadores:** Son símbolos utilizados para realizar operaciones matemáticas, de asignación o lógicas. Algunos ejemplos son "+", "-", "*", "/", "=", "&&", "||", entre otros. Destacan los operadores booleanos.
- **Delimitadores:** Son caracteres utilizados para separar y agrupar diferentes elementos en el código. Algunos ejemplos son ";", "{", "}", "(", ")", "[", "]", entre otros.
- **Comentarios:** Son textos utilizados para explicar o documentar el código. No afectan al funcionamiento del programa, pero ayudan a entender el código.

- **Palabras clave:** Son palabras que tienen un significado específico en un lenguaje de programación y se utilizan para realizar acciones específicas. Por ejemplo, "import", "public", "private", "return", entre otras.

Ejemplo en C#:

```
// Declaración de la función
public static void Saludar(string nombre)
{
   Console.WriteLine("Hola, " + nombre);
}

// Llamada a la función
Saludar("Juan");
```

En este ejemplo, se declara una función llamada Saludar que recibe un parámetro de tipo string llamado nombre. La función utiliza el método Console.WriteLine para imprimir un mensaje de saludo en la consola. Luego, se llama a la función pasando el argumento "Juan". (educaopen tech school, 2025)

2.6 Funciones y librerías básicas: Las funciones y librerías básicas son bloques de código predefinidos que se pueden utilizar para realizar tareas comunes.

¿Cómo se utilizan las librerías de programación?

Cada librería está diseñada para proporcionar una solución a una necesidad o un problema de desarrollo específico. Esto puede incluir autenticación de usuario, conexión al servidor, interfaces de usuario, gestión de datos, algoritmos, animaciones, etc. Así, los desarrolladores a menudo buscan librerías para un componente de software específico que quieren crear rápidamente o con el que tienen problemas.

Una vez identificada la librería que puede resolver su problema o necesidad, desde el software que están desarrollando utilizan llamadas a la función o funciones concretas de la librería que necesitan y se realiza una tarea completa que, de lo contrario, requeriría un código de varias docenas de líneas.

Las funciones de una librería son los fragmentos de código que a partir de unos parámetros de entrada (llamada a la función), generan una salida o resultado concreto. Por ejemplo, al pasarle las credenciales de un usuario (nombre y contraseña) lo autentican en la aplicación, para que tenga acceso y permisos para utilizarla.

Todos los lenguajes de programación admiten librerías externas y hay innumerables para distintos propósitos y funciones.

Tipos de librerías de programación

Existen librerías para todos los lenguajes de programación que se utilizan en el desarrollo de *software*. A continuación, se comentarán algunos tipos de librerías ampliamente extendidas en programación:

- **Fecha y hora:** para tareas relacionadas con la fecha y la hora, como obtener la fecha actual, la diferencia entre dos fechas…
- **Sistema operativo y sistema:** estos tipos de librerías se utilizan para interactuar con el sistema operativo sobre el que se desarrolla o un sistema del *software*. Entre sus funcionalidades figuran obtener información sobre el sistema, rutas de archivos, directorios, etc.
- ***Web-scraping*:** para extraer datos de una web en un formato legible, conectarse a un sitio web de forma segura, finalizar la conexión cuando sea necesario…
- **Desarrollo de GUI:** para crear interfaces gráficas de usuario de las aplicaciones. Son las pantallas con las que interactúan los usuarios de las aplicaciones.

- **Desarrollo de juegos:** se trata de librerías muy especializadas. Se utilizan para construcción de animación en 2D y 3D, creación de *sprites*, construcción de mundos, etc. Este tipo de librerías son unas de las más difíciles de dominar.
- **Programación web:** para actividades relacionadas con la web: conectarse a una web, aplicación o servicio web, programación de *sockets*, etc. (unir, 2025)

2.7 Documentación de programas: La documentación de programas es el conjunto de comentarios y descripciones que se agregan al código para explicar su funcionamiento y propósito.

¿Qué es la documentación en programación?

En programación, la documentación es más que una reflexión; es un aspecto esencial del desarrollo de software. Pero, ¿qué *es* exactamente la documentación en programación? En términos simples, es el texto escrito o las ilustraciones que acompañan al software o al código, explicando cómo funciona, cómo usarlo y por qué se tomaron ciertas decisiones durante el desarrollo. Sirve como guía para desarrolladores, usuarios y partes interesadas, asegurando que todos estén en la misma página.

Importancia de la documentación de software en el SDLC

El Ciclo de Vida del Desarrollo de Software (SDLC) es un proceso estructurado que incluye varias etapas, desde la planificación y diseño hasta las pruebas y el mantenimiento. La documentación juega un papel crítico a lo largo de estas etapas, actuando como un mapa que guía a los equipos a través del desarrollo y más allá. Sin una documentación adecuada, incluso el código bien escrito puede volverse incomprensible, lo que lleva a aumentar los costos de mantenimiento, retrasar proyectos y frustrar a los desarrolladores.

Entendiendo la documentación de software de computadoras

Definición y propósito

La documentación de software de computadora es una colección completa de información que detalla la funcionalidad, arquitectura y uso del software. Su propósito principal es asegurar que el software pueda ser entendido, utilizado y mantenido por diversos interesados, incluidos desarrolladores, probadores, usuarios y futuros mantenedores.

Componentes clave de una documentación efectiva

La documentación efectiva es clara, concisa y bien organizada. Típicamente incluye:

- Introducción: Proporciona una visión general del software, su propósito y su alcance.
- Guías de usuario: Instrucciones paso a paso sobre cómo usar el software.
- Documentación de API: Información detallada sobre cómo interactuar con el software de forma programática.
- Comentarios de código: Explicaciones en línea dentro de la base de código, aclarando lógicas o decisiones complejas.
- Diagramas y visuales: Ayudas visuales como diagramas de flujo y diagramas que ayudan a comprender la estructura y el flujo de datos del software.

Tipos de documentación de software

Documentación de requisitos

Este tipo de documentación captura los requisitos funcionales y no funcionales del software. Actúa como un contrato entre las partes interesadas y los desarrolladores, delineando lo que el software debe hacer y las limitaciones que debe cumplir.

Documentación de arquitectura/diseño

La documentación de arquitectura o diseño proporciona un plano de la estructura del software, detallando los componentes de alto nivel, sus interacciones y los patrones de diseño subyacentes. Es crucial para la incorporación de nuevos desarrolladores y para mantener la consistencia en proyectos grandes.

Documentación técnica

La documentación técnica está dirigida a desarrolladores y usuarios técnicos, ofreciendo detalles profundos sobre los entresijos del software. Esto incluye documentación de API, instrucciones de configuración y guías de implementación.

Documentación de usuario

La documentación de usuario está diseñada para los usuarios finales, explicando cómo instalar, configurar y usar el software. Esto puede variar desde manuales simples hasta sistemas de ayuda interactivos incorporados en el software.

Documentación de API

La documentación de API es una forma especializada de documentación técnica que proporciona detalles sobre cómo interactuar con la API del software. Incluye descripciones de métodos, parámetros de entrada, formatos de salida y ejemplos. (getguru, 2025)

2.8 Descripción de estructuras de datos utilizadas: Las estructuras de datos son formas de organizar y almacenar datos en un programa.

¿Qué es una Estructura de Datos?

De manera simple, una estructura de datos es un contenedor que almacena datos en una disposición específica. Esta "disposición" permite que una estructura de datos sea eficiente en algunas operaciones e ineficiente en otras. Tu meta es

comprender las estructuras de datos para que puedas elegir la que sea más óptima para el problema en cuestión.

¿Por qué necesitamos Estructuras de Datos?

Así como las estructuras de datos son usadas para almacenar datos de una forma organizada, y dado que los datos son la entidad más crucial en informática, el verdadero valor de las estructuras de datos es claro.

No importa qué problema estés resolviendo, de un modo u otro tienes que tratar con datos — ya sea el salario de un empleado, precios de acciones, una lista de compras, o incluso un directorio telefónico simple.

Basado en diferentes escenarios, los datos necesitan ser almacenados en un formato específico. Tenemos un puñado de estructuras de datos que cubren nuestra necesidad de almacenar datos en distintos formatos.

Estructuras de Datos más Usadas

Enlistemos primero las estructuras de datos más usadas, y después las revisaremos una por una:

1. Arreglos
2. Pilas
3. Colas
4. Listas Enlazadas
5. Árboles
6. Grafos
7. Tries (en realidad son árboles, pero es bueno mencionarlos por separado).
8. Tablas Hash

(FreeCodeCamp, 2023)

2.9 Código fuente: El código fuente es el texto que se escribe en un lenguaje de programación para crear un programa.

¿Qué es el código fuente?

Los ordenadores, ya sean PC domésticos, modernos móviles u ordenadores para la industria o la ciencia, trabajan **con un sistema binario:** encendido/apagado, cargado/no cargado, 1/0. Una secuencia de estados (bits) indica al ordenador lo que tiene que hacer. Mientras que en los comienzos de la tecnología informática se creaban comandos con estas dos condiciones, hace tiempo que se ha pasado a escribir aplicaciones en un lenguaje de programación legible por los humanos. Esto puede sonar un poco raro *a priori*, pues para los profanos en la materia un código fuente es más un galimatías que otra cosa. En este contexto, "legible por los humanos" es antónimo de "**legible por las máquinas** ". Mientras que los ordenadores solo trabajan con valores numéricos, las personas utilizamos palabras para comunicarnos. Al igual que las lenguas extranjeras, los lenguajes de programación también tienen que aprenderse antes de utilizarse.

Diversos lenguajes de programación

Hay cientos de lenguajes de programación diferentes y no se puede decir *pese* que unos sean mejores que otros, pues esto depende del contexto del proyecto y de la aplicación para la que se use el código fuente. Entre los **lenguajes de programación más conocidos** se encuentran:

- BASIC
- Java
- C
- C++
- Pascal

- Python
- PHP
- JavaScript

Para que los ordenadores puedan comprenderlos, estos deben traducirse al código de máquina.

Compilador e intérprete

Para que los ordenadores puedan procesar el texto fuente creado por los programadores tiene que haber un **traductor** entre ambos en forma de programa adicional. Esta aplicación auxiliar puede presentarse como compilador o como intérprete:

- **Compilador**: este tipo de aplicación traduce (compila) el código fuente en un código que el procesador puede comprender y ejecutar. Este código de máquina se almacena en forma de archivo ejecutable.

- **Intérprete**: un intérprete traduce el código fuente línea a línea y lo ejecuta directamente. El proceso de traducción funciona mucho más rápido que en un compilador, pero la ejecución es más lenta y se necesita una gran cantidad de memoria.

Con todo, la elección de uno u otro no es libre, pues es el el lenguaje de programación el que determina si debe utilizarse un compilador o un intérprete, aunque hoy en día cada vez es más frecuente recurrir a una solución provisional: **just-in-time** (**JIT**) **compilation,** en español compilación en tiempo de ejecución. Este tipo de traducción intenta combinar las ventajas de ambos programas (análisis y ejecución rápidos) y se emplea, por ejemplo, en los navegadores para gestionar JavaScript, PHP o Java más eficazmente.

Lenguajes de marcado

El texto fuente también es el término utilizado para definir la estructura básica de una página web. Sin embargo, esta no se basa en un lenguaje de programación, sino en el lenguaje de marcado HTML. Un lenguaje de marcado establece la manera en que se estructuran los contenidos. Así, por ejemplo, HTML permite definir encabezados, párrafos o resaltes. Un documento HTML no es en sí un programa, pero puede incluir alguno en forma de código JavaScript, lo que también se aplica a otros lenguajes de marcado como XML.

Estructura del texto fuente

Al escribir programas hay que cumplir determinadas convenciones independientemente del lenguaje de programación que se emplee. Muy pocos lenguajes se crean de la nada, sino que la mayoría se desarrolla a partir de los otros, de ahí que haya determinados elementos que aparecen reiteradamente en diferentes códigos de programación:

- **Comandos**: las instrucciones son probablemente la base de todas las aplicaciones. Con ellas, los programadores indican a sus futuros programas qué es lo que tienen que hacer. Tales comandos pueden, por ejemplo, desencadenar determinados procesos de cálculo o incluso mostrar un texto.

- **Variables**: las variables son espacios en los que se insertan datos. Dentro del código fuente se hace referencia a estas una y otra vez mediante la asignación de un nombre.

- **Comparaciones**: especialmente decisivas para la estructura de la mayoría de programas son las consultas que funcionan según un esquema causa-efecto,

es decir, siguiendo el principio de la lógica proposicional. La introducción de un valor lógico determinado desencadena un evento; si no, se produce uno diferente.

- **Bucles**: las consultas también pueden constituir la base para los bucles del texto fuente. Un comando se repite hasta que se haya alcanzado un valor determinado, tras lo cual el programa abandonará el bucle y ejecutará el resto del código.
- **Comentarios**: todos los lenguajes de programación actuales permiten comentar líneas dentro del código, con lo que es posible escribir texto en el código fuente que el programa no tiene en cuenta. El autor puede así introducir comentarios en el texto fuente para que él mismo u otro desarrollador puedan entender en el futuro las diferentes partes del código.

La creación de código fuente siempre está ligada a un problema. Los desarrolladores escriben programas para **ofrecer soluciones**, pero no se establece cuál es el camino para ello. Cuando dos programadores se ocupan del mismo problema, puede ocurrir que ambos textos fuente difieran significativamente entre sí, incluso a pesar de que se trabaje con el mismo lenguaje.

A pesar de que en muchos casos no hay una solución única, todas las tareas de programación tienen algo en común: un buen texto fuente debe prescindir de código innecesario, pues este hace que el programa sea más complejo, lento y propenso a errores. El código fuente poco claro que ni siquiera los profesionales pueden comprender se denomina **código espagueti**, pues su estructura es tan confusa como un montón de espaguetis en un plato.

Creación del código fuente

Para escribir un texto fuente solo se necesita un editor de textos simple, como Editor (cuyo nombre original es Notepad) en Windows o TextEdit en Mac. El código fuente se guarda como texto sin formato (por ejemplo en código ASCII o como UTF-8) a partir de la extensión del nombre de archivo correcto para el lenguaje de programación. Si encuentras un archivo con la terminación “.cpp” en tu disco duro, se trata de un **archivo de texto** que contiene código en el lenguaje de programación C++. (IONOS, 2020)