Módulo 1. Programación Lógica. Algoritmos y lenguajes. Lenguaje Java

Bienvenido al curso introductorio de programación. Este curso cuenta con apuntes en formato de lecturas que incrementan la complejidad, especialmente diseñadas para un aprendizaje cómodo que sigue un lineamiento pedagógico ideal de competencias y aula invertida. El mismo fue desarrollado por el magíster Walter F. Agüero, docente de varias universidades argentinas que vierten conceptos propios en todo el material y que significan un resumen de varias bibliografías leídas por el autor. Cuando amerite una cita bibliográfica, la misma será nombrada en formato APA para que el alumno pueda ampliar la lectura.

1. Historia de la computación y paradigmas

Antes de comenzar a desarrollar este punto, es necesario tener en cuenta que en la actualidad un equipo informático está formado por *hardware* y *software*. El *hardware* es la parte física, la que se puede tocar en un equipo, y el *software* es lo intangible, es el *software*. Hoy en día, el crecimiento va de la mano, teniendo un futuro prometedor con la tecnología *quanum* y el *software* que puede ejecutarse en él. Como en toda historia, es necesario empezar desde el principio para conocer el presente y orientarnos hacia donde vamos.

La historia de la computación abarca desde antiguas herramientas de cálculo como el ábaco hasta las modernas computadoras, pasando por las máquinas analíticas de Charles Babbage (siglo XIX), consideradas el primer concepto de computadora. En el siglo XX, con la invención del transistor y el desarrollo de la electrónica, surgieron las computadoras modernas. Las primeras computadoras electrónicas como el ENIAC, el desarrollo de lenguajes de alto nivel y, finalmente, la era de los microprocesadores y las PC. Los paradigmas computacionales, o formas de pensar la computación, han evolucionado desde lo mecánico hasta lo algorítmico, lo de programación estructurada, orientada a objetos, lógica y el desarrollo de inteligencia artificial.

Orígenes y antecedentes

Antigüedad

El ábaco, inventado hace miles de años, es considerado el primer computador mecánico para cálculos. La siguiente figura muestra qué es un ábaco.

Figura 1: Ábaco

Fuente: [imagen sin título sobre ábaco]. (s. f.). https://bit.ly/3K3CTom

---

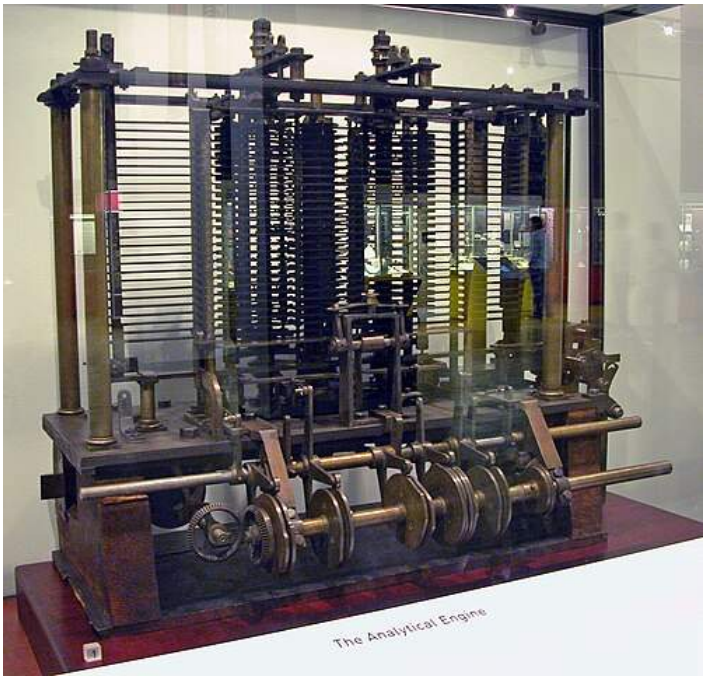
Siglo XVII

Se desarrollan las primeras máquinas de calcular mecánicas por inventores como Blaise Pascal y Gottfried Leibniz.

Siglo XIX

Charles Babbage diseña la máquina analítica, considerada la primera computadora de propósito general, con memoria y capacidad de ejecutar algoritmos, aunque no se llegó a construir completamente. Ada Lovelace escribe el primer algoritmo para esta máquina. En la siguiente figura, podremos observar la máquina analítica.

Figura 2: Máquina analítica



Fuente: [imagen sin título sobre máquina analítica]. (s. f.). https://bit.ly/4i9ZUIZ

La Era Moderna y los primeros ordenadores electrónicos

Años 30 y 40

Se establecen las bases teóricas de la computación con Alan Turing y su concepto de máquina de Turing, que define la idea de «computabilidad». Este gran científico tiene una presencia actual que se verá más adelante.

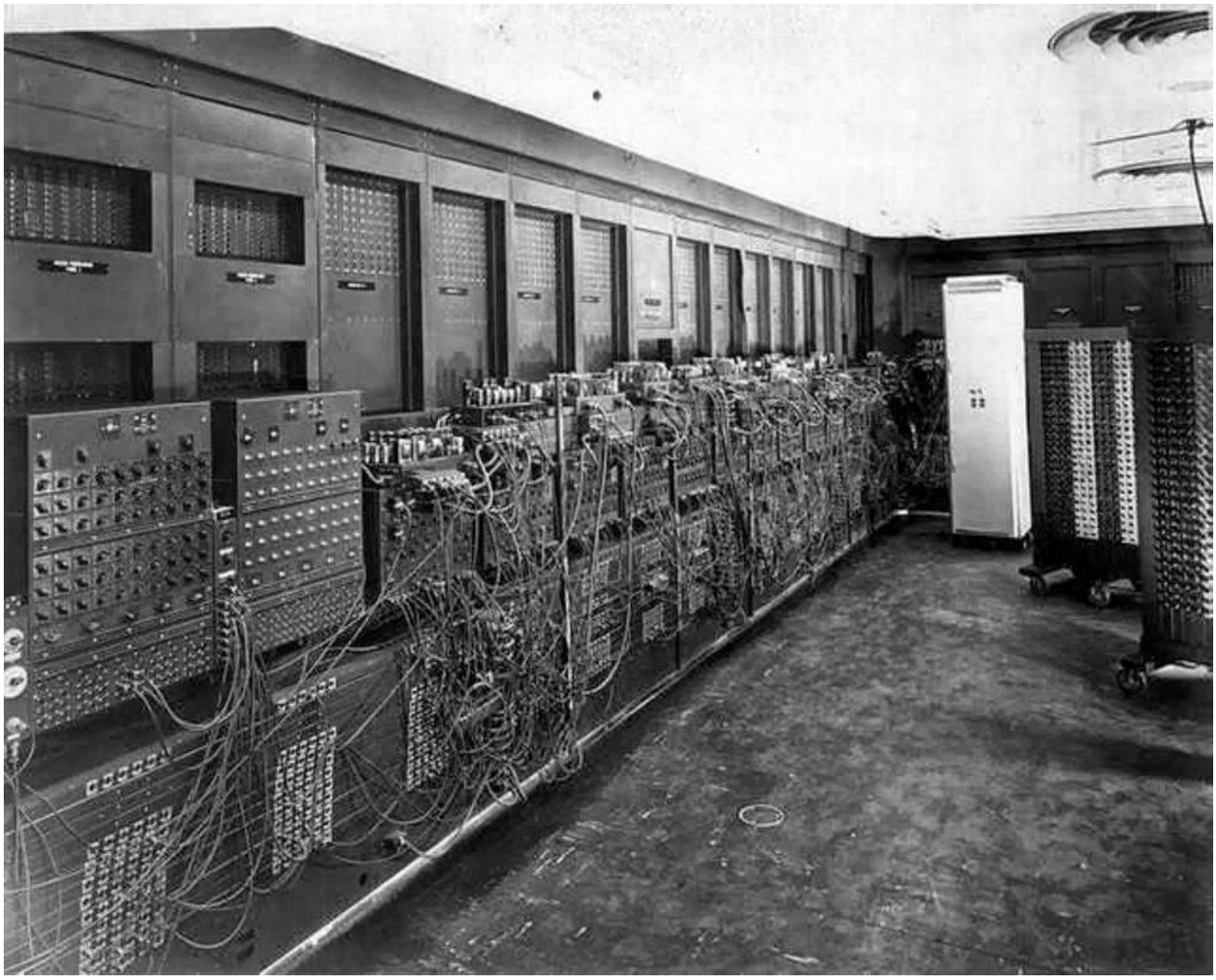
Figura 3: Alan Turing

Fuente: [imagen sin título sobre Alan Turing]. (s. f.). https://bit.ly/481yom8

Años 40

Se construyen los primeros ordenadores electrónicos programables. El ENIAC, en 1946, fue el primero, y requería reprogramación manual a través de cables.

Figura 4: Primeros ordenadores electrónicos programables



Fuente: [imagen sin título sobre primeros ordenadores electrónicos programables]. (s. f.). https://bit.ly/4oQlzm1

Años 50

Se crea el primer lenguaje ensamblador, por Kathlyn Wood, para facilitar la programación. Comienza la segunda generación de computadoras, reemplazando las válvulas de vacío por transistores.

Desarrollo de la computación y los lenguajes

AÑOS 50 Y 60	AÑOS 70	AÑOS 80

Nacen los lenguajes de programación de alto nivel como Fortran y COBOL. Se desarrollan los primeros circuitos integrados, y la investigación en inteligencia artificial toma fuerza.

AÑOS 50 Y 60	AÑOS 70	AÑOS 80

Se desarrolla el lenguaje C, un hito en la programación de sistemas, y se inventan lenguajes de programación lógica como Prolog.

AÑOS 50 Y 60	AÑOS 70	AÑOS 80

Se vuelven comunes las computadoras personales (PC), con la IBM PC, de 1981, estableciendo estándares de la industria.

Paradigmas de programación

Los paradigmas son diferentes enfoques o estilos para resolver problemas de computación y diseños de *software*.

Figura 5: Paradigmas de lenguajes

Paradigma	Lenguaje	Características	Ventajas	Desventajas	Año de Creación	Autor	Soporta Datos Masivos	Usa Tecnología Cuántica
Cuántico	Q#	Diseñado para algoritmos cuánticos, integración con Azure Quantum.	Optimizado para computación cuántica.	Uso limitado en hardware real actual.	2017	Microsoft	Sí	**Sí**
Cuántico / Funcional	Quipper	Diseñado para describir algoritmos cuánticos.	Adecuado para computación cuántica a gran escala.	Uso limitado fuera del ámbito cuántico.	2013	Peter Selinger et al.	Sí	**Sí**
Orientado a Objetos	Swift	Lenguaje moderno de Apple para iOS y macOS.	Seguro, rápido, sintaxis moderna.	Enfocado al ecosistema Apple.	2014	Apple Inc.	No	**No**
	Java	Encapsulamiento, herencia y polimorfismo.	Modularidad, reutilización de código.	Mayor complejidad que los lenguajes procedurales.	1995	James Gosling	Sí	**No**
Orientado a Objetos / Funcional	Kotlin	Moderno, conciso, interoperable con Java.	Seguro, fácil de mantener, ideal para desarrollo Android.	Menor comunidad que Java.	2011	JetBrains	Sí	**No**
Multiparadigma	Julia	Alto rendimiento, tipado dinámico, orientado a datos científicos.	Rápido, ideal para datos masivos y simulaciones científicas.	Ecosistema más joven que Python o R.	2012	Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman	Sí	**No**
	Rust	Seguridad de memoria, sin recolector de basura.	Seguridad y rendimiento.	Curva de aprendizaje.	2010	Graydon Hoare	No	**No**
	Go	Compilado, concurrente, eficiente.	Simplicidad, rendimiento, ideal para servidores.	Falta de genéricos (hasta versiones recientes).	2009	Robert Griesemer, Rob Pike, Ken Thompson	Sí	**No**
	Scala	Combina programación funcional y orientada a objetos.	Compatible con Java, ideal para big data (Apache Spark).	Curva de aprendizaje.	2003	Martin Odersky	Sí	**No**
	R	Especializado en análisis estadístico y visualización de datos.	Ideal para análisis de datos, amplio soporte estadístico.	Menor rendimiento comparado con otros lenguajes.	1993	Ross Ihaka y Robert Gentleman	Sí	**No**
	Python	Admite orientación a objetos, programación funcional e imperativa.	Sintaxis simple, gran comunidad, bibliotecas extensas.	Menor rendimiento en comparación con C/C++.	1991	Guido van Rossum	Sí	**Sí**
	C++	Soporta programación orientada a objetos, genérica e imperativa.	Rendimiento, control sobre el hardware.	Complejidad del lenguaje.	1985	Bjarne Stroustrup	No	**No**
Funcional	Elixir	Concurrente, funcional, basado en Erlang VM.	Escalabilidad, tolerancia a fallos.	Curva de aprendizaje moderada.	2011	José Valim	Sí	**No**
	Haskell	Evaluación perezosa, funciones puras, sin estado mutable.	Facilita pruebas y depuración.	Curva de aprendizaje pronunciada.	1990	Comité de académicos	No	**No**
	Erlang	Concorrencia extrema, tolerancia a fallos.	Escalabilidad masiva, ideal para sistemas distribuidos.	Sintaxis diferente, comunidad más pequeña.	1986	Joe Armstrong, Ericsson	Sí	**No**
	Lisp	Listas como estructura principal, macros, recursión.	Muy flexible, base para muchos lenguajes de IA.	Sintaxis poco convencional.	1958	John McCarthy	No	**No**

Fuente: elaboración propia.

Programación imperativa —

Se centra en el «cómo» se hacen las cosas, con secuencias de instrucciones que cambian el estado de un programa.

Programación orientada a objetos —

Organiza el *software* en «objetos» que contienen datos y funciones, fomentando el uso de la reutilización de código.

Programación lógica —

Se basa en la lógica formal, donde el programador expresa las relaciones y hechos, y el sistema busca una solución (por ejemplo: Prolog).

Programación funcional —

Trata la computación como la evaluación de funciones matemáticas, evitando el cambio de estado y la mutabilidad de los datos.

Inteligencia artificial

Uno de los paradigmas más avanzados busca que las computadoras puedan resolver problemas de forma más natural y autónoma. Los paradigmas de programación son enfoques para resolver problemas computacionales; a continuación, los desarrollaremos.

- **Programación imperativa**: basada en instrucciones secuenciales.
- **Programación orientada a objetos**: organiza el código en objetos que encapsulan datos y comportamientos.
- **Programación funcional**: se basa en funciones matemáticas sin efectos secundarios.
- **Programación lógica**: utiliza reglas lógicas para derivar conclusiones.

CONTINUAR

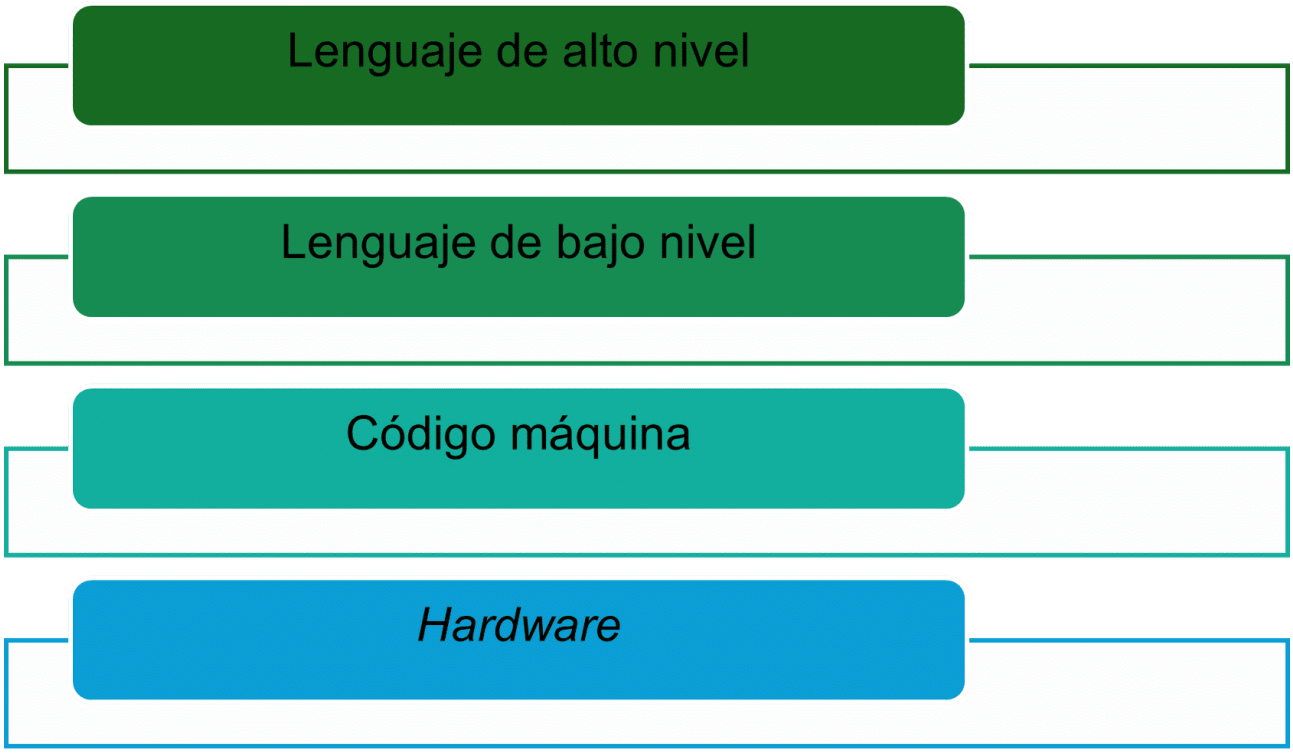
2. Lenguaje máquina, ensamblador, lenguajes de alto y bajo nivel

El lenguaje máquina es el conjunto de instrucciones que una computadora puede ejecutar directamente, representadas en código binario.

El lenguaje ensamblador es una representación simbólica del lenguaje máquina, más comprensible para los humanos. Requiere un ensamblador para traducirse a código máquina.

Los lenguajes de bajo nivel (como ensamblador) están cerca del *hardware* y ofrecen control detallado. Los lenguajes de alto nivel (como Java, Python) son más abstractos y fáciles de usar, permitiendo escribir código más legible y portable.

Figura 6: Lenguajes de bajo y alto nivel



Fuente: elaboración propia.

Compiladores e intérpretes

Compilador

Un compilador traduce todo el código fuente de un programa a código máquina antes de ejecutarlo.

Ejemplos: GCC para C/C++, Javac para Java.

Un intérprete traduce y ejecuta el código línea por línea en tiempo real. Ejemplos: Python.

JavaScript

Ventajas del compilador

- Mayor velocidad de ejecución.
- Detección de errores antes de ejecutar.

Ventajas del intérprete

- Facilidad para probar y depurar.
- Portabilidad entre plataformas.

CONTINUAR

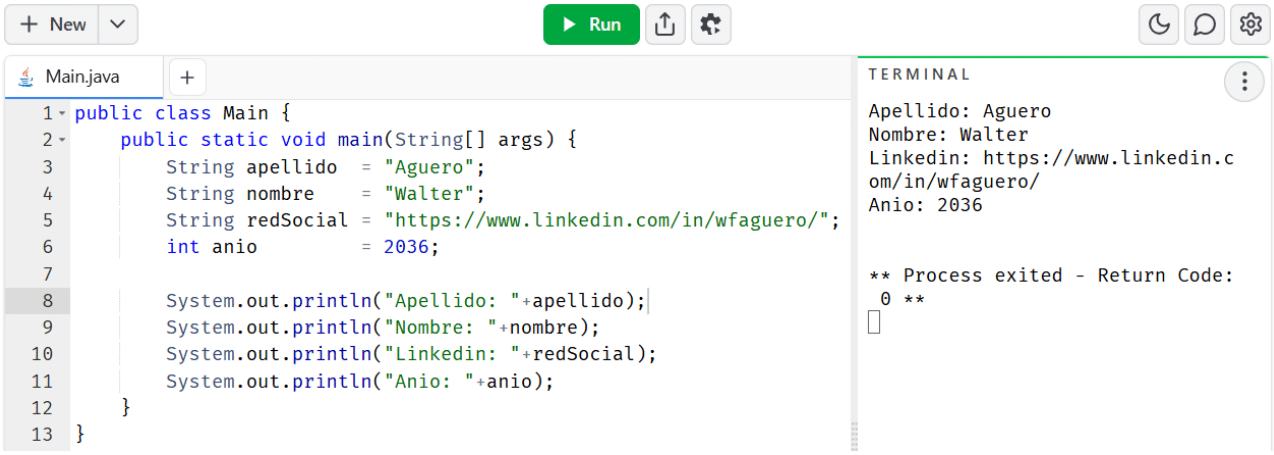
3. Historia de Java

Java fue desarrollado por Sun Microsystems en 1995. Fue diseñado para ser un lenguaje de programación orientado a objetos, seguro, portátil y con capacidad de ejecución en múltiples plataformas gracias a la máquina virtual de Java (JVM).

Características clave

- Sintaxis similar a C/C++.
- Recolección automática de basura.
- Multiplataforma: «*write once*, *run anywhere*».
- Amplio uso en aplicaciones empresariales, móviles (Android), y web.

Figura 7: Código lenguaje Java



Fuente: Elaboración propia.

Para ver y ejecutar el código fuente, haga clic en el siguiente enlace: https://www.online-java.com/Y8VUCYefXI

Estructura de un programa en Java

Un programa en Java se compone de clases. La clase principal contiene el método «*main*», punto de entrada del programa.

A continuación, veremos un ejemplo.

```
public class HolaMundo {

 public static void main(String[] args) {

  System.out.println(«¡Hola Mundo!»);

 }

}
```

A continuación, mencionaremos los elementos clave.

- Clase: estructura principal.
- Método main: ejecutado al iniciar el programa.
- System.out.println: imprime texto en consola.

CONTINUAR

4. Tipos de datos y operadores

Es importante conocer los tipos de datos, ya que ayudará a entender su alcance cuando se explique el concepto de atributos en el lenguaje Java. Si pensamos que un atributo es semejante a un *locker* donde, al igual que los *locker* de cheques bancarios, solo es posible guardar cheques, los atributos pueden guardar únicamente valores definidos por su tipo de datos y no podrán alojar otro. Por ejemplo, si define un atributo de tipo entero, no podrá guardar números decimales, solo números enteros.

Java tiene tipos de datos primitivos, como veremos a continuación.

- int: números enteros. Por ejemplo: -10, 20, 50, etc.
- double: números decimales. Por ejemplo: 1,45, 0,45, etc.
- char: caracteres. Por ejemplo: «a», «b», «A», «@», etc.
- boolean: verdadero o falso.

A continuación, mencionaremos los operadores comunes.

- Aritméticos: +, -, *, /, %.
- Relacionales: ==, !=, >, <, >=, <=.

- Lógicos: &&, ||, !

A continuación, veremos ejemplos.

1. int a = 5; atributo cuyo nombre es a, y que está definido como entero (int).
2. int b = 3.
3. int suma = a + b; // suma = 8 (el atributo suma, es de tipo entero y aloja la suma del contenido del atributo a, más el contenido del atributo b).
4. boolean resultado = (a > b) && (b > 0); // true.

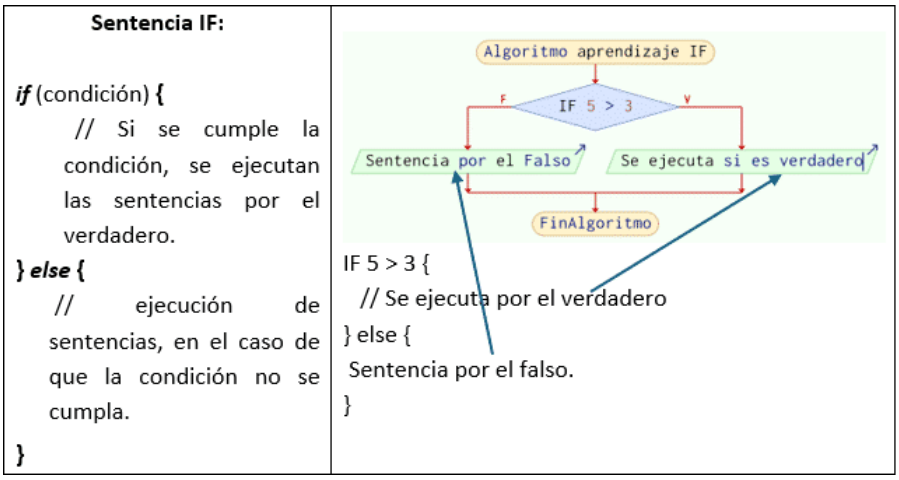
CONTINUAR

5. Estructuras de control

Las estructuras de control permiten modificar el flujo de ejecución del programa.

Condicionales (IF): si la condición es verdadera, se ejecutan la/s sentencias por el verdadero; si no, se ejecutan la/s sentencias por el falso.

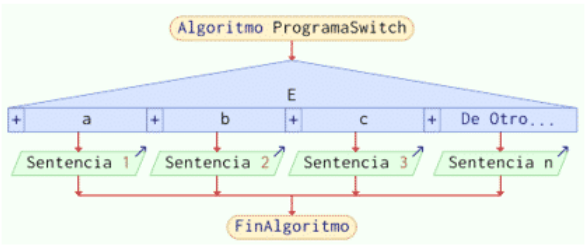
Figura 8: Algoritmo aprendizaje IF



Fuente: elaboración propia.

Switch: evalúa la variable y consulta cuál es de igual el valor (*case*) y se ejecutan las sentencias. En caso de no coincidir si tiene *default*, se ejecutan estas.

Figura 9: Algoritmo programa *switch*

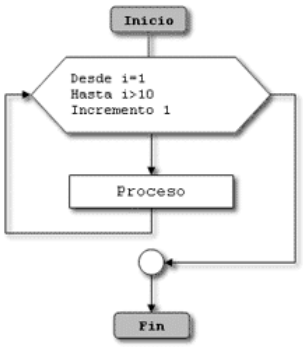
switch (variable) {
 	
case valor1:
 	
// código
 	
break;
 	
default:
 	
// código
 	
}
	
	
	Switch **(**E**) {**
	Case a: Sentencia 1; otras sentencias...; *break*;
	Case b: Sentencia 2; otras sentencias...; *break*;
	Case c: Sentencia 1; otras sentencias...; *break*;
	Default: Sentencias... n; *break*;
	}

Fuente: elaboración propia.

Bucles

For: se inicializa la variable por única vez, y ejecuta la condición. Si es verdadera, ejecuta todas las sentencias encerradas por la llave que contiene el *for* y, al final, incrementa el contador y vuelve a consultar la condición.

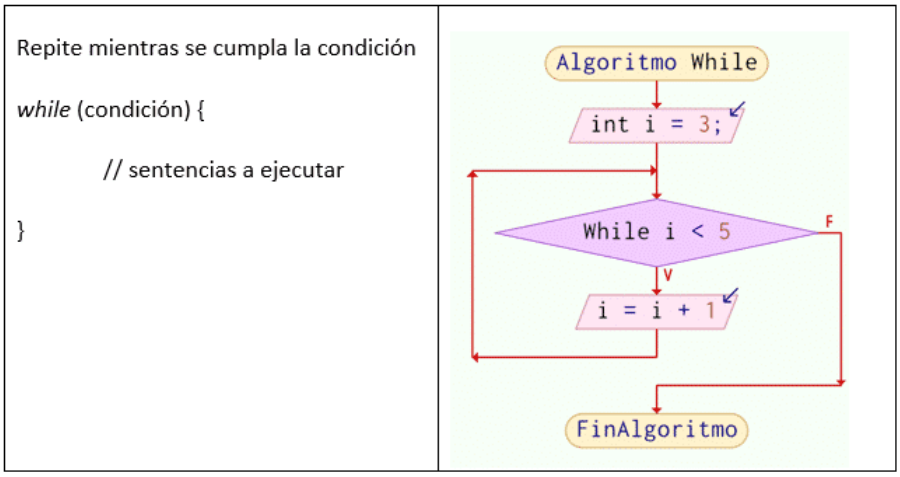
Figura 10: Bucles

For (inicialización, condición, incremento) {
Sentencias
}
 	
Ejemplo:
 	
for (int i = 0; i < 10; i++) {
// sentencias
}
	

Fuente: [imagen sin título sobre bucles]. (s. f.). https://bit.ly/4824Rsr

While: si la condición se cumple, itera o repite las sentencias hasta que sea falsa. Aquí la sentencia se puede ejecutar cero o más veces.

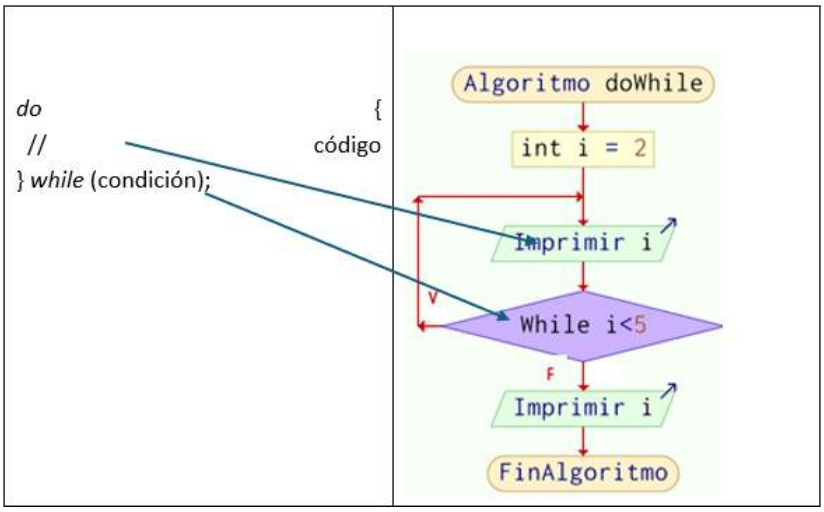
Figura 11: Algoritmo *while*



Fuente: elaboración propia.

Do...While: se ejecutan las sentencias mientras sea verdadera la condición. Aquí al menos una vez se ejecutará la sentencia que se repite o itera.

Figura 12: Algoritmo *dowhile*



Fuente: elaboración propia.

CONTINUAR

6. IDE de programación

Un IDE (entorno de desarrollo integrado) es una aplicación de *software* que unifica las herramientas necesarias para desarrollar código de manera eficiente. Reúne un editor de código con resaltado de sintaxis, un depurador para encontrar errores, y herramientas de compilación que automatizan tareas repetitivas, todo en una sola interfaz gráfica. Al integrar estas funcionalidades, un IDE aumenta la productividad del programador al simplificar el proceso de creación, edición, prueba y empaquetado de *software*.

Características principales de un IDE

Editor de código: permite escribir código de forma más eficaz, ofreciendo resaltado de sintaxis para mejorar la legibilidad, autocompletado de código y detección de errores en tiempo real.

Depurador (*debugger*): una herramienta que ayuda a los desarrolladores a identificar y solucionar errores en su código, permitiendo probar el programa paso a paso.

Automatización de compilaciones: incluye herramientas que automatizan tareas repetitivas del proceso de desarrollo, como la compilación del código fuente y la ejecución de pruebas.

Gestión de proyectos: proporciona una interfaz para organizar archivos, directorios y dependencias del proyecto de manera intuitiva.

Beneficios de usar un IDE

Mayor productividad: al integrar varias herramientas en una sola interfaz, se agiliza el flujo de trabajo y se ahorra tiempo.

Mejor calidad del código: las funciones como el autocompletado y la detección de errores en tiempo real ayudan a los programadores a escribir código más limpio y sin fallos.

Reducción de la complejidad: simplifica el proceso de desarrollo al evitar la necesidad de usar múltiples programas por separado para cada tarea.

Ejemplos de IDE populares

- Visual Studio: un IDE muy popular de Microsoft, utilizado para el desarrollo de aplicaciones .NET y otros lenguajes.
- PyCharm: especializado en el desarrollo de aplicaciones en el lenguaje Python.
- Eclipse: ampliamente usado en programación Java y compatible con otros lenguajes.
- Xcode: el IDE de Apple para el desarrollo de aplicaciones en iOS y macOS.
- Visual Studio Code: un editor de código ligero y extensible, muy popular para el desarrollo web.

Figura 13: IDE para programar

Fuente: Juan Villalvazo (16 de febrero de 2017). https://bit.ly/4o7Wpy6

CONTINUAR

7. IDE ONLINE

Un IDE en línea u «*online* IDE» (entorno de desarrollo integrado en línea) es una aplicación web que permite a los programadores escribir, probar y depurar código desde un navegador web, sin necesidad de instalar *software* localmente. Ofrece un conjunto de herramientas para el desarrollo de *software*, como un editor de código, compiladores y herramientas de gestión de control de versiones, todo accesible a través de internet para facilitar la colaboración y el acceso desde cualquier lugar.

Características clave de un IDE en línea

Acceso basado en navegador

Puedes codificar, construir y colaborar directamente desde tu navegador web, eliminando la necesidad de configuraciones locales.

Entorno consistente

Proporcionan entornos de desarrollo estandarizados y controlados, lo que garantiza que los desarrolladores trabajen con herramientas uniformes.

Colaboración en tiempo real

Muchas plataformas permiten a múltiples usuarios trabajar juntos en el mismo proyecto simultáneamente.

Gestión de control de código fuente

Suelen integrarse con sistemas como Git, facilitando la gestión de versiones y el seguimiento de cambios.

Herramientas integradas

Incluyen un editor de código avanzado con resaltado de sintaxis, herramientas de compilación y depuración.

Acceso universal —

Permiten a los desarrolladores acceder a su trabajo y herramientas desde cualquier dispositivo con conexión a internet.

Ejemplos de IDE en línea

- **GitHub *Codespaces*:** ofrece un entorno de desarrollo completo y personalizado directamente en la nube, integrado con el ecosistema de GitHub.
- **GitHub *Codespaces*:** ofrece un entorno de desarrollo completo y personalizado directamente en la nube, integrado con el ecosistema de GitHub.
- **GitLab web IDE:** proporciona un entorno de desarrollo integrado dentro de la interfaz de GitLab, con capacidades para editar código, gestionar archivos y realizar *commits*.

Beneficios

- **Productividad:** aumenta la productividad del desarrollador al centralizar las herramientas necesarias para el desarrollo.
- **Facilita la incorporación:** los nuevos miembros de un equipo pueden empezar a trabajar rápidamente sin pasar por una compleja instalación de *software*.

- **Colaboración:** facilita el trabajo en equipo, especialmente en proyectos distribuidos o remotos.

Para este curso se usará una IDE *online* que, al hacer clic, aparecerá el código fuente del programa que se envíe como ejemplo y los alumnos además podrán compartir su código fuente de la misma manera.

Usaremos en este curso la IDE *online*: https://www.online-java.com/

Si hacemos clic en el enlace, se nos abrirá en nuestro navegador la IDE mencionada.

En la siguiente figura, se modificaron algunas líneas del código inicial que se muestra cuando se abre por primera vez la IDE para que el alumno pueda observar cómo, a partir de un código inicial, puede modificar y luego ejecutar presionando el botón en verde (RUN).

Figura 14: IDE – *online Java*

Fuente: elaboración propia.

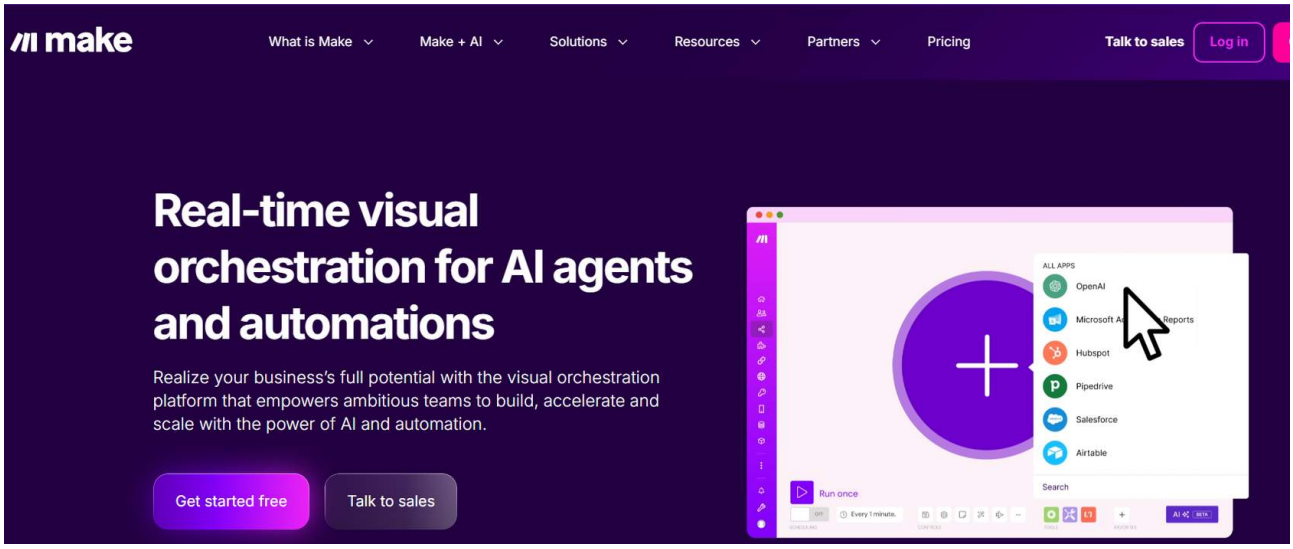
En la figura anterior se puede ver que hay palabras en color azul; esto indica que son palabras reservadas, es decir, no se pueden usar en otro lugar que no sea el permitido, como en este caso. Sobre el mismo, el alumno

irá aprendiendo a lo largo del curso. En la pantalla a la derecha indica TERMINAL: aquí muestra el resultado de ejecutar el programa; en este caso, muestra dos mensajes.

Make.com

En próximas lecturas se incorporará el concepto de Make que se anticipa seguidamente

Figura 15: Make



Fuente: captura de pantalla de Make (Make.com).

CONTINUAR

Referencias

[Imagen sin título sobre ábaco]. (s. f.). https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcQEDnTgMsoUDp-4bN7syVreywPGuEC1yMEtvrrXrID1l18uoVc9

[Imagen sin título sobre Alan Turing]. (s. f.). https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRYO79diEeWeJ8z1G6LhBYpenelCi7S1HcPTCNmPzRHYzqb-aRY

[Imagen sin título sobre bucles]. (s. f.). https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcQZg3qPTlYZ34KMLytsDC284egy56Jbinj1QMH-osVialD3JUle

[Imagen sin título sobre máquina analítica]. (s. f.). https://upload.wikimedia.org/wikipedia/commons/thumb/a/ac/AnalyticalMachine_Babbage_London.jpg/1200px-AnalyticalMachine_Babbage_London.jpg

[Imagen sin título sobre primeros ordenadores electrónicos programables]. (s. f.). https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQaJsPV5HTlYutsPvDw8qXFW_DsCfte-cYxxZM3IJz-CyL7lAXV

Juan Villalvazo (16 de febrero de 2017). ¿Qué es un IDE de Programación? Significado de Entorno de Desarrollo Integrado. [Archivo de Video]. YouTube. https://www.youtube.com/watch?v=_WKWpJEv9UY

CONTINUAR