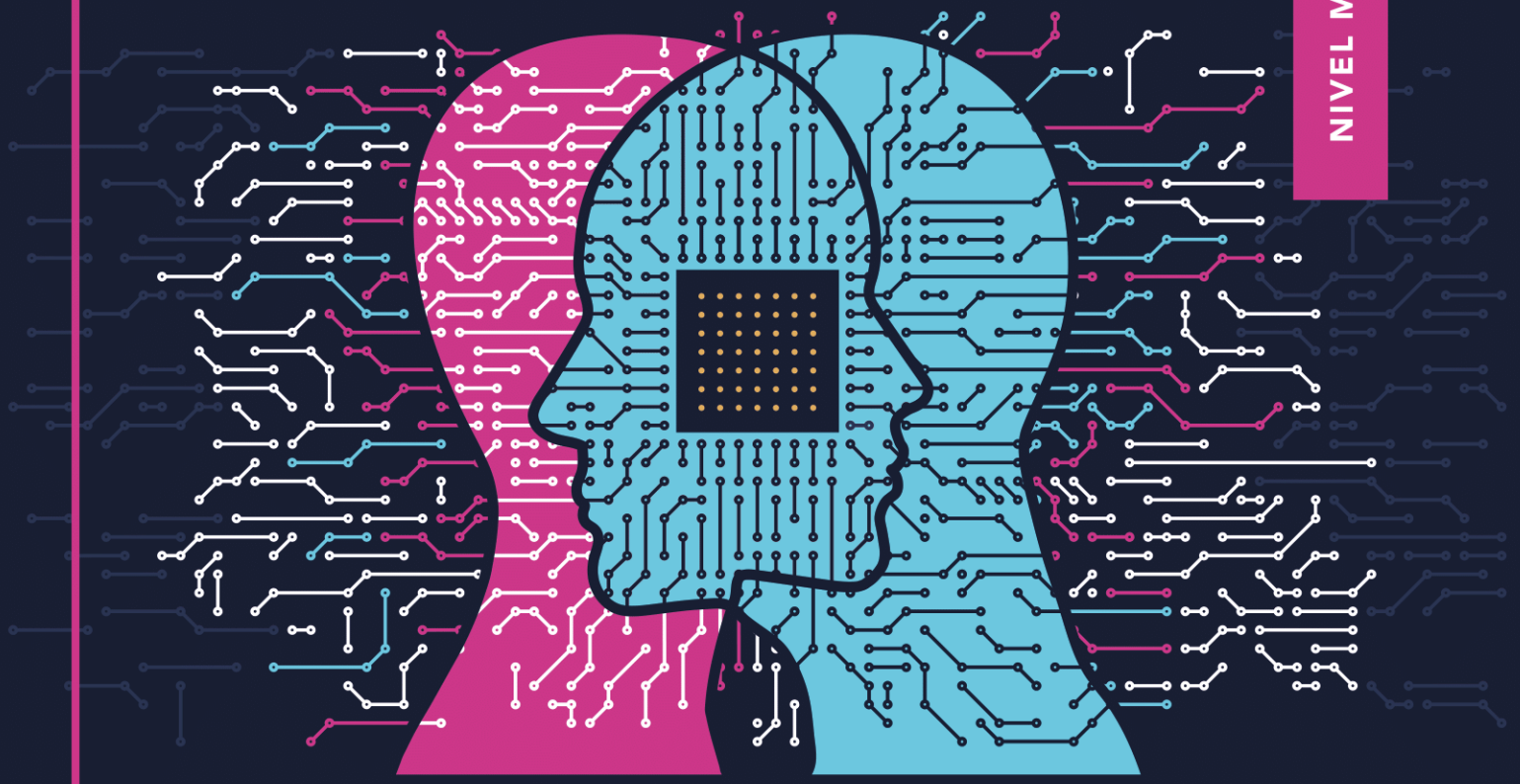
PENSAMIENTO COMPUTACIONAL

NIVEL MEDIO SUPERIOR

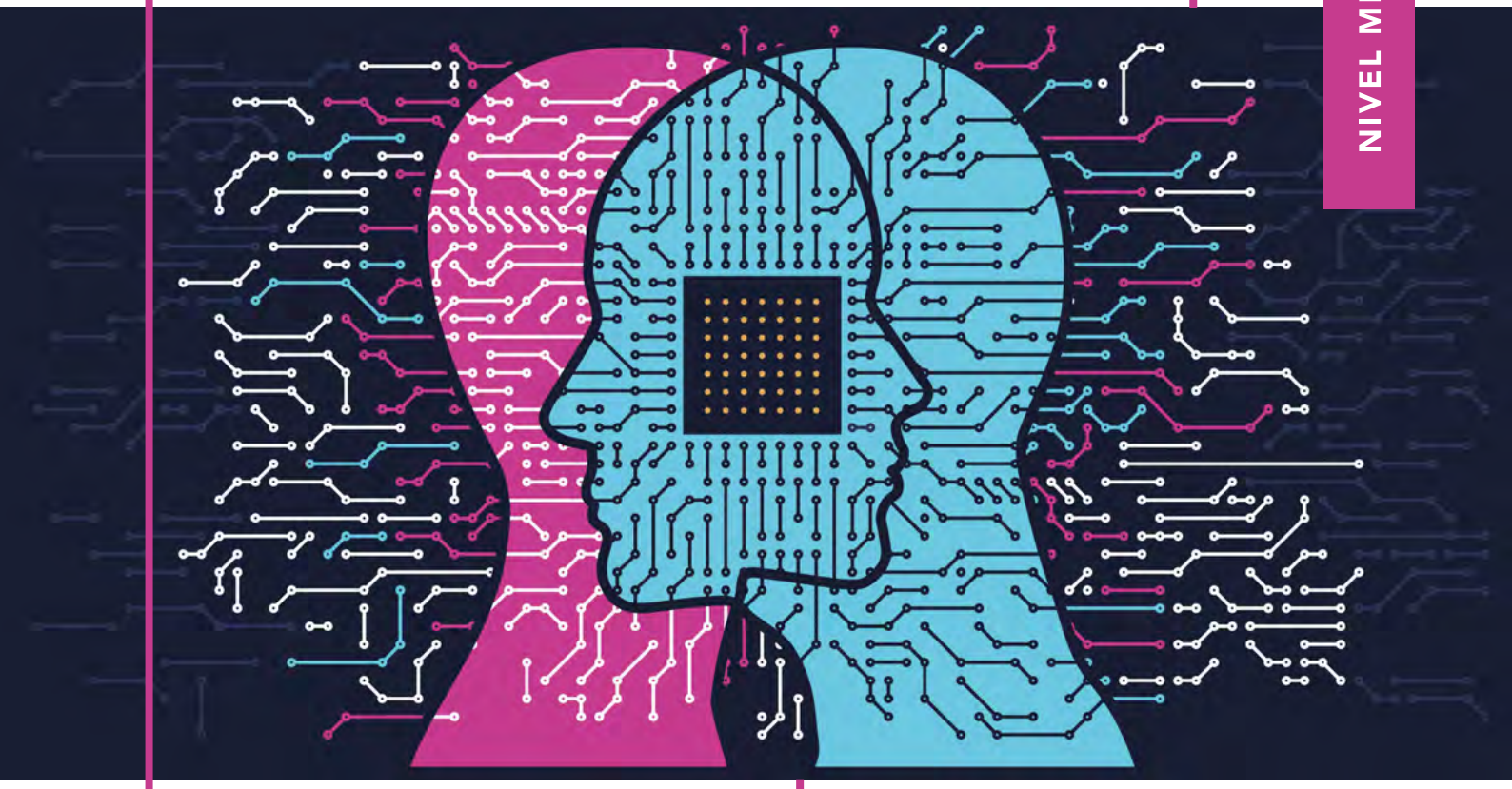


Claudia De Anda Quintin
Edwin Ramón Romero Espíritu
Gibrán Uriel López Coronel
Rigoberto Santiago Garzón

GYROS EDITORIAL

PENSAMIENTO COMPUTACIONAL

NIVEL MEDIO SUPERIOR



Claudia De Anda Quintin
Edwin Ramón Romero Espíritu
Gibrán Uriel López Coronel
Rigoberto Santiago Garzón

GYROS EDITORIAL

Director Editorial y Producción:
Gustavo González Gallina

Director Administrativo:
Irma Vega Doñez

Diseño y diagramación:
Departamento de Arte y diseño GYROS

Foto de portada:
Shutterstock

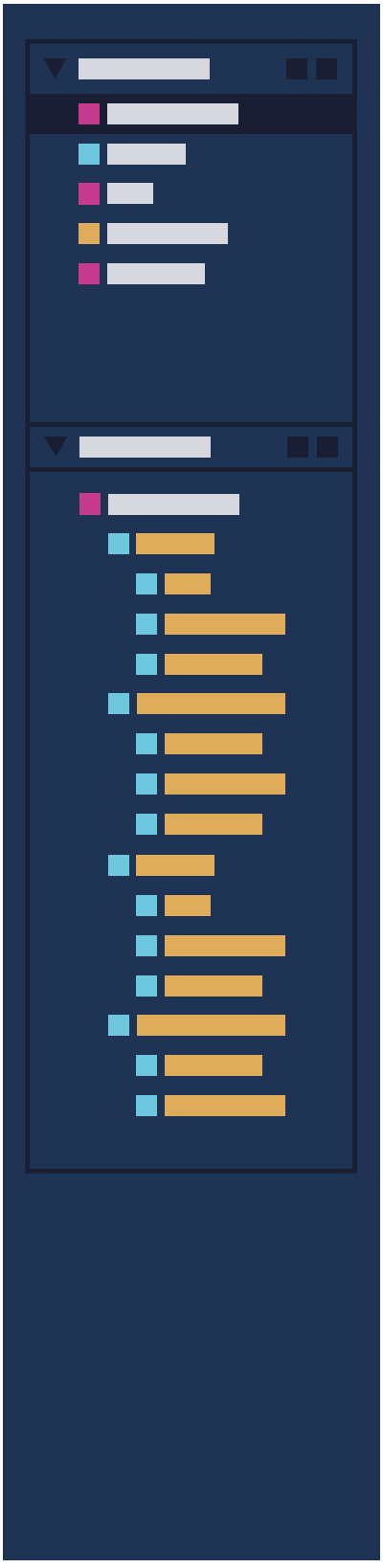
Título de la obra: Pensamiento Computacional
Primera edición 2026

Pensamiento Computacional
Primera edición 2026

ISBN: 978-970-96930-4-1

Impreso en Monterrey, México
Impresión 2026

Presentación



El libro **Pensamiento computacional** se construyó de acuerdo con los lineamientos didáctico-pedagógicos del Programa de estudios de la unidad de aprendizaje curricular del mismo nombre del Plan de estudios Bachillerato UAS 2024, emitido por la Dirección General de Escuelas Preparatorias de la Universidad Autónoma de Sinaloa.

El Programa de estudios en mención, se orienta con los enfoques humanista y constructivista del Modelo educativo UAS 2022 y con los lineamientos de la Nueva Escuela Mexicana, que buscan la construcción de una sociedad con fundamento en el humanismo y en la ciencia; además de orientarte hacia el desempeño idóneo en los diversos contextos culturales y sociales, hacerte protagonista de tu propio proceso de aprendizaje partiendo del desarrollo y fortalecimiento de tus habilidades cognoscitivas y metacognitivas e incorporarte a la Educación Superior o al mundo laboral. Asimismo, se enfatizan las estrategias didácticas oportunas para que adquieras conocimientos y experiencias acordes a las exigencias presentes y futuras, derivadas de los rápidos cambios tecnológicos que transforman a la sociedad, haciendo imprescindible dotarte, en la medida de lo posible, de habilidades tecnológicas y de la utilización de herramientas digitales, que te faciliten el acceso y el análisis de información, y que te permiten comunicar, divulgar, socializar, modelar, crear, simular, manipular, interactuar e investigar.

En ese sentido, los principios pedagógicos de los contenidos del presente título se alinean con un enfoque educativo colaborativo, adaptable a las realidades y contextos, además promueven un aprendizaje activo y reflexivo planteado a través de metodologías activas y participativas, basabas en la indagación y el descubrimiento de conocimientos en pro de que desarrolles capacidades analíticas, críticas y reflexivas.

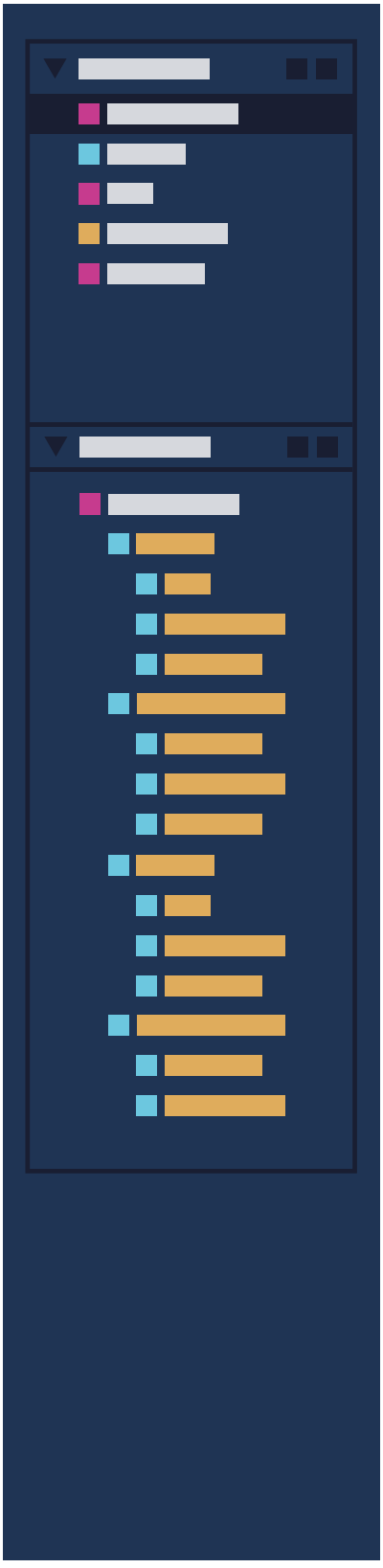
Los contenidos de la obra se diseñan bajo un modelo que desarrollarás progresivamente y te guiarán al logro de las metas, de manera que el desarrollo de tus habilidades y la construcción de tu aprendizaje se plantean trabajarlas en cinco progresiones, a través de las cuales identificarás e implementarás las fases del pensamiento computacional resolviendo problemas cotidianos y académicos mediante la construcción de algoritmos en un entorno de desarrollo integrado y el lenguaje de programación estructurada C++, utilizando estructuras decisivas e iterativas para automatizar procesos, manipular conjuntos de datos, validar la solución de manera digital y determinar la ejecución de instrucciones de manera organizada y eficiente. Además, simularás sistemas robóticos mediante aplicaciones gráficas y la programación en Arduino con funciones elementales, control de salidas y el uso de sensores y actuadores para resolver problemas simples de automatización. Para cumplir con estos propósitos académicos no bastará el conocimiento y la comprensión de los conceptos expuestos en esta obra, sino también en que resuelvas actividades que te llevarán a la reflexión y autoanálisis, para que examines tu propio proceso de aprendizaje, revises tus fortalezas y debilidades vividas durante el proceso de aprendizaje y así transformar y mejorar tu vida y el entorno social, económico y profesional en el que te desarrollas.

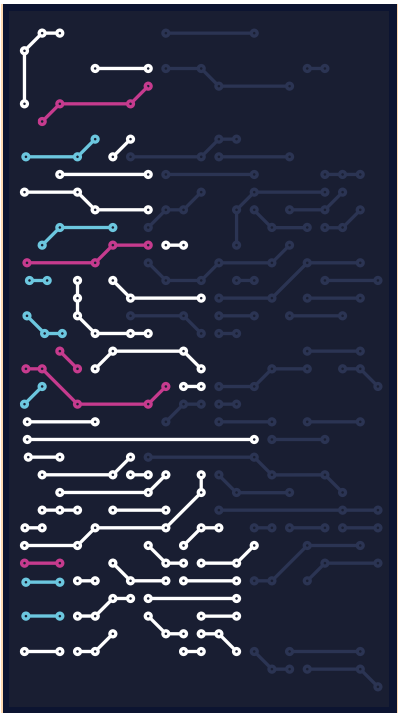
Agradecimientos

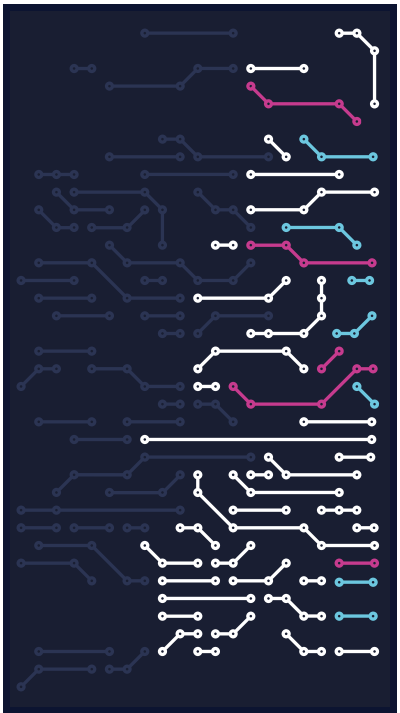
Nuestro sincero reconocimiento a los docentes integrantes del cuerpo colegiado de la disciplina de Informática de la Dirección General de Escuelas Preparatorias de la Universidad Autónoma de Sinaloa, quienes colaboraron en la elaboración de recursos didácticos para este libro de texto.

Gracias, colegas por compartir con la comunidad educativa y con cada generación de estudiantes del Bachillerato universitario, sus conocimientos, creatividad y experiencia, plasmados en este recurso didáctico.

Ángel Sánchez Díaz
Eduin Alejandro Laveaga Corrales
Eva Angelina Martínez Campaña
Francisco Eduardo Aispuro García
Frida Bibiana Ñonthe Ortiz
Gabriela Avendaño Sainz
Jesús Alfredo Ramírez Aviña
Jesús González Aldaz
Jesús Ignacio Hernández García
Jesús Miguel Almeida Muñoz
Luis Alfredo Ramírez Aviña
Mariela Lilián García Ramos
Nadya Rocío Galaviz Heredia
Oscar Urías Fierro
Raquel Villa Núñez
Rosario Garnica Núñez
Sabby Carolina Hernández Gárate



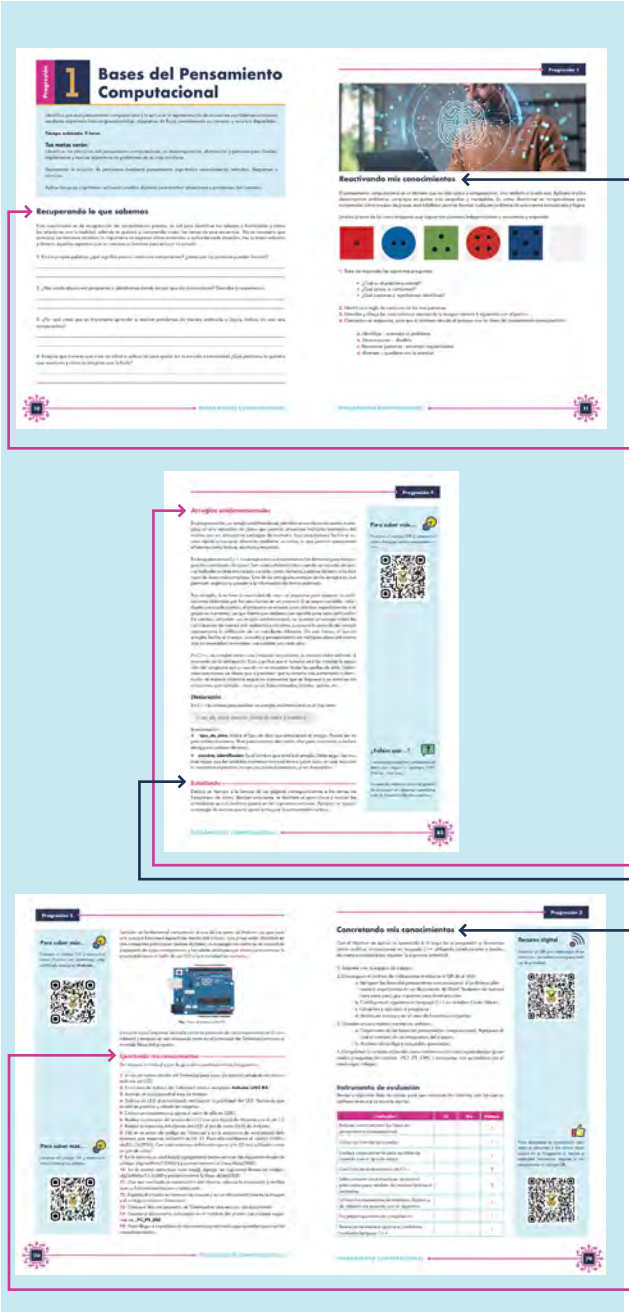




Conoce tu Libro

El libro **Pensamiento computacional**, ha sido diseñado como recurso didáctico para la asignatura del mismo nombre, la cual está inserta en el cuarto semestre del mapa curricular del Plan Bachillerato UAS 2024 de la Universidad Autónoma de Sinaloa.

La obra está conformada por cinco secuencias didácticas que progresivamente abordarán los temas ayudándote en la integración de saberes y el desarrollo de tus habilidades. Cada una de ellas está constituida por contenidos y diferentes tipos de actividades de aprendizaje, dispuestas para que adquieras y apliques tus conocimientos; asimismo evidencies el desempeño y el nivel de logro de las metas enmarcadas en los aprendizajes de trayectoria del programa de estudios de la asignatura.



Los componentes del libro son:

Entrada de la secuencia

En esta sección se presenta la progresión de aprendizaje que será abordada en la secuencia y las metas a lograr en el trayecto.

Recuperando lo que sabemos. Es un cuestionario de evaluación diagnóstica que debes responder antes de abordar cada progresión de aprendizaje, es útil para que recuperes tus saberes y reconozcas tus fortalezas acerca de los temas que estudiarás en cada secuencia. Este tipo de actividad no representa una valoración numérica en tu evaluación.

Secuencia por progresión

Reactivando mis conocimientos. Al inicio de cada secuencia didáctica de las progresiones, se presenta una situación o problemática con preguntas que te guiarán a relacionar tus conocimientos previos con los temas a estudiar.

Desarrollo del tema. Es el apartado que contiene el discurso escrito de los temas y las actividades que te ayudarán a trabajar de manera individual y colaborativa en el desarrollo de tus habilidades y a poner en práctica tus saberes. En el desarrollo se incluyen secciones y cápsulas que te permitirán descubrir tus actitudes y manifestarlas en la evaluación.

Tipos de actividades

Estudiando. En algunas ocasiones va a ser necesario que realices actividades fuera de clase, que te ayudarán a prepararte para el tema que se abordará o que refuerces lo practicado. Es muy importante que atiendas las indicaciones y realices las tareas.

Ejercitando mis conocimientos. Este tipo de actividades refieren a prácticas a desarrollarse durante las clases, en el centro de cómputo con la guía del profesor. Su ponderación representa un alto porcentaje en tu evaluación.

Concretando mis conocimientos. Son actividades de aprendizaje interrelacionadas y orientadas para que las trabajes de manera autónoma. Están diseñadas para que realices procedimientos que te encaminan a evidenciar el nivel de logro de las metas

propuestas en cada progresión. Al finalizar cada progresión encontrarás una actividad de este tipo. También tienen asignado un alto valor en la evaluación, por lo que es importante atender la retroalimentación que te haga el profesor, mejores la evidencia de acuerdo con las observaciones hechas y reenvíes para su revaloración.

Actividades alternativas. A lo largo del curso, en tres momentos distintos, encontrarás estas actividades que son complementarias o de recuperación, en su mayoría son propuestas que derivan de la retroalimentación. En el caso que tengas interés de mejorar tu evaluación, puedes solicitar al profesor que te indique en qué momento realizarlas.

▶ Valorando mi aprendizaje

Reflexionando lo que aprendí. Como parte de la evaluación metacognitiva, en tres momentos del curso, se te solicitará respondas algunas preguntas que implica reflexiones acerca de tu propio proceso de aprendizaje, para concretar los conocimientos y seas consciente de ello. No representan una valoración en tu evaluación final, por lo que puedes responderlas lo más sincero posible.

Autoevaluación. En el apartado de Valorando lo que aprendí, encontrarás instrumentos que te ayudarán a medir tu nivel de dominio de los aspectos de aprendizaje de las metas. Son útiles para ayudarte a regular tu aprendizaje, te indicarán cuáles ajustes necesitas hacer para reforzar lo aprendido.

Coevaluación. La evaluación entre pares ayuda en el proceso de aprendizaje colaborativo, por lo que en este libro se integran instrumentos para que evalúes el desempeño general de tu equipo de trabajo durante el desarrollo de las actividades de aprendizaje colaborativas.

▶ Cápsulas

Conceptos clave. Son empleadas para definir conceptos que es importante domines para comprender los temas.

Relaciónalo con.... Describe información más profunda del tema para que establezcas su vínculo con otras unidades de aprendizaje curricular, con tu vida cotidiana o tu comunidad.

Para saber más. Con estas cápsulas se accede a videotutoriales, presentaciones interactivas, infografías, entre otros, para ampliar alguna explicación del tema en cuestión.

¿Sabías qué...? Son cápsulas con información adicional, interesante o datos curiosos que actualizarán tu aprendizaje en torno a las herramientas digitales.

Recurso digital. Se incluyen en algunas secciones del libro y están referidos a recursos didácticos como formatos y plantillas con indicaciones o cuestionarios interactivos, útiles para que evidencies tu aprendizaje.

Progresión 1

Bases del Pensamiento Computacional

Identifica qué es el pensamiento computacional y lo aplica en la representación de soluciones a problemas cotidianos mediante algoritmos básicos (pseudocódigo, diagramas de flujo), considerando su contexto y recursos disponibles.

Tiempo estimado: 9 horas

Tus metas serán:

- Identificar los principios del pensamiento computacional, su descomposición, abstracción y patrones para diseñar, implementar y evaluar algoritmos de problemas de su vida cotidiana.
- Representar la solución de problemas mediante pensamiento algorítmico seleccionando métodos, diagramas o técnicas.
- Aplicar lenguaje algorítmico utilizando medios digitales para resolver situaciones o problemas del contexto.

Recuperando lo que sabemos

Este cuestionario es de recuperación de conocimientos previos, es útil para identificar tus saberes y habilidades y cómo los relacionas con la realidad, además te ayudará a comprender mejor los temas de esta secuencia. No es necesario que conozcas los términos técnicos; lo importante es expresar cómo entiendes o aplicarías cada situación, haz tu mejor esfuerzo y detecta aquellos aspectos que no conoces o dominas para enfocar tu estudio. .

1. En tus propias palabras ¿qué significa pensar como una computadora? ¿crees que las personas pueden hacerlo?

_____

_____

2. ¿Has usado alguna vez programas o plataformas donde tengas que dar instrucciones? Describe tu experiencia.

_____

_____

3. ¿Por qué crees que es importante aprender a resolver problemas de manera ordenada o lógica, incluso sin usar una computadora?

_____

_____

4. Imagina que tuvieras que crear un robot o aplicación para ayudar en tu escuela o comunidad ¿Qué problema te gustaría que resolviera y cómo te imaginas que lo haría?

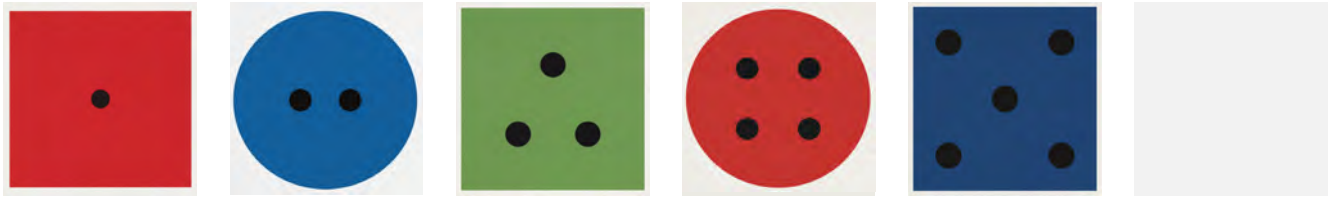
_____

_____

Reactivando mis conocimientos

El pensamiento computacional es un término que no sólo aplica a computadoras, sino también a la vida real. Aplicarlo implica descomponer problemas complejos en partes más pequeñas y manejables. Es como desmontar un rompecabezas para comprender cómo encajan las piezas, esta habilidad permite abordar cualquier problema de una manera estructurada y lógica.

Analiza la serie de las cinco imágenes que siguen tres patrones independientes y recurrentes:



1. Trata de responder las siguientes preguntas:

- ¿Cuál es el problema central?
- ¿Qué tareas lo componen?
- ¿Qué patrones o repeticiones identificas?

2. Identifica la regla de cada uno de los tres patrones.
3. Describe y dibuja las características exactas de la imagen número 6 siguiendo con el patrón.
4. Compartan su respuesta, para que el profesor vincule el proceso con las fases del pensamiento computacional:

a. Identificar – entender el problema
b. Descomponer – dividirlo
c. Reconocer patrones - encontrar regularidades
d. Abstraer – quedarse con lo esencial

.

1.1 Inicios de la algoritmia

Relaciónalo con...

Estudios y revisiones han señalado que el Pensamiento computacional mejora la capacidad de resolución de problemas cuando se integra de forma articulada en educación primaria y secundaria. En México, la SEP ha comenzado a integrarlo en educación básica y media superior para promover habilidades como el razonamiento lógico, la creatividad, la toma de decisiones y el trabajo colaborativo, desafíos del siglo XXI.

En estos tiempos, donde la tecnología transforma cada aspecto de nuestras vidas, el pensamiento computacional se ha convertido en una competencia esencial para cualquier persona que enfrenta problemas complejos en su vida diaria o profesional. Esta habilidad permite a los estudiantes abordar cualquier problema complejo de manera lógica, estructurada y eficiente, utilizando herramientas propias de la informática, pero aplicables a cualquier disciplina.

Contrario a lo que se puede creer el pensamiento computacional no se limita a la programación, es una forma de pensar que implica descomponer problemas, reconocer regularidades, abstraer lo esencial y formular pasos claros, es decir, algoritmos que conduzcan a soluciones repetibles y verificables. Formalmente se puede decir que el **Pensamiento computacional** es un conjunto de procesos cognitivos y estrategias para formular, analizar y resolver problemas de manera que puedan ser resueltos por una persona, una computadora o una combinación de ambos.

Orígenes del pensamiento computacional

La columna vertebral del pensamiento computacional es el algoritmo, término que asociamos con la informática y que ha sido una práctica milenaria usada para describir pasos al resolver problemas.

Un **algoritmo** es un conjunto finito y ordenado de instrucciones, pasos o reglas bien definidas, que permite solucionar un problema. Sus características clave son:

Preciso	Ordenado	Finito	Definido	Concreto
Con pasos planteados de forma objetiva y sin ambiguedades.	Secuencia debe ser clara y precisa.	Tener un número determinado de pasos.	Con mismos datos producir el mismo resultado.	Ofrecer una solución especifica.

La algoritmia o el arte de diseñar algoritmos ha acompañado a la humanidad desde que se empezó a sistematizar la resolución de problemas. Se tiene a los babilonios y los Sumerios, quienes desde el año 3000 a,C. utilizaban técnicas algorítmicas

para realizar cálculos en tablillas de arcilla, como multiplicaciones o la estimación de raíces cuadradas. En la antigua Grecia el matemático Euclides describió el famoso *Algoritmo de Euclides*, un procedimiento para encontrar el Máximo Común Divisor (MCD) de dos números.

El salto a la Era de la computación

En el siglo XX se dio la formalización de algoritmos y en el siglo XXI los expertos en educación propusieron enseñar el pensamiento computacional desde edades tempranas, considerándola una habilidad tan fundamental como leer y escribir. Hoy día se exploran formas de integrarlo con otras disciplinas bajo el enfoque STEAM (Ciencia, Tecnología, Ingeniería, Arte y Matemáticas).

El concepto de la algoritmia se fusionó con la informática gracias a piones visionarios como:

- Ada Lovelace, quien es considerada la primera programadora de la historia al trabajar con la Máquina Analítica de Charles Babbage, en 1842 escribió lo que hoy se conoce como el **primer algoritmo informático**, diseñado para que la máquina calculara la secuencia de números de Bernoulli.
- George Boole desarrolló el Algebra de Boole o **Algebra Booleana**, un sistema que describe el pensamiento lógico usando solos dos valores: verdadero y falso (0/1). Este es el fundamento lógico de toda la programación y la codificación digital actual.
- Alan Turing, en 1936 formalizó el concepto teórico de algoritmo con su modelo de la **Maquina de Turing**, una abstracción matemática de una computadora que puede ejecutar cualquier algoritmo. Durante la Segunda Guerra Mundial con su trabajo se descifraron códigos, esto fue clave en descomposición de problemas y el diseño algorítmico.
- John Von Neumann propuso la **Arquitectura Von Neumann** en el 1945. Este modelo permite a las computadoras almacenar en la misma memoria tanto el programa (el algoritmo) como los datos, importante para los algoritmos ejecutables.

Desde estos cimientos, la algoritmia ha evolucionado hasta convertirse en la base de la Inteligencia Artificial y otras aplicaciones usadas diariamente.

Relación entre algoritmo y pensamiento computacional

La formalización del pensamiento lógico y algorítmico consolidó la genealogía histórica y las múltiples dimensiones del pensamiento computacional, con los métodos computacionales, ingeniería de software, ciencias computacionales y diseño.

De manera que **el pensamiento computacional enseña a pensar en términos algorítmicos** (estructura, repetición, condiciones, modularidad), pero también en capacidades previas como formular el problema y evaluar resultados. Así pues el diseño del algoritmo es una de las practicas centrales del pensamiento computacional, dado que implica traducir una solución conceptual a pasos claros, verificables y si es posible, ejecutables por una computadora.

Para saber más...

Accede al video Inicios de la algoritmia, para ampliar la explicación del tema. Hazlo escaneando el Código QR.

¿Sabías qué...?

Existen algoritmos que imitan procesos naturales como Algoritmos genéticos, basados en la evolución biológica; Colonia de hormigas, para encontrar rutas óptimas; Algoritmos de enjambre de partículas, inspirados en el comportamiento de aves o peces. Todos estos se usan en inteligencia artificial, robótica y optimización.

1.2 Fases del pensamiento computacional

Relaciónalo con...

La Segunda Guerra Mundial fue un punto de inflexión para la computación moderna, Turing y su equipo en Bletchley Park desarrollaron técnicas computacionales avanzadas para descifrar mensajes cifrados, un proceso que implicaba descomposición de problemas, reconocimiento de patrones y diseño de algoritmos; todas fases del pensamiento computacional.

Recurso digital

Escanea el QR para acceder a la infografía Estrategias de comprensión lectora.

Entender mejor el mundo digital que nos rodea, por ejemplo cómo funcionan las redes sociales, cómo se procesan los datos personales o por qué cada aplicación actúa de cierta manera ayuda a tomar decisiones informadas y seguras, además, la sociedad demanda adaptabilidad a los constantes cambios de las formas de producir e interactuar. Ante este contexto dinámico, el pensamiento computacional provee herramientas cognitivas que ayuden a analizar contextos de forma inmediata, evaluar diferentes escenarios y tomar decisiones informadas para automatizar procesos repetitivos.

El pensamiento computacional no es un paso único, sino un proceso cíclico y estructurado que consta de varias etapas interconectadas que facilitan la resolución de problemas:

- **Identificación del problema**
- **Descomposición**
- **Reconocimiento de patrones**
- **Abstracción**
- **Diseño de algoritmo**
- **Implementación**
- **Evaluación**

Estas fases no son lineales, sino que forman un ciclo constante donde la evaluación y el refinamiento pueden llevar a reajustar la descomposición, la abstracción o el diseño del algoritmo.

Estudiando

Dedica un tiempo a la lectura de las páginas correspondientes a los temas de **Fases del pensamiento computacional**. Realizar esta tarea, te facilitará el aprendizaje y realizar las actividades que el profesor guiará en las siguientes sesiones.

Apóyate en alguna estrategia de lectura que te ayude a mejorar la comprensión lectora. Con el recurso digital de al lado puedes conocer algunas.

Identificación del problema

Esta primera fase es crucial, conlleva formular de la manera más precisa posible para que las herramientas de tecnología puedan ayudar a encontrar la solución.

Por tanto, antes de buscar una solución es imprescindible comprender los requisitos, las limitaciones y los objetivos del problema en cuestión, entender el desafío a descifrar. Una buena identificación separa hechos de supuestos y enuncia criterios de éxito medibles.

Componentes de la fase de identificación

- **Análisis del problema.** Se examina el problema en profundidad para entender todos sus aspectos. Esto implica identificar las necesidades, restricciones y objetivos que debe cumplir la solución.

- **Definición de los elementos.** Se especifican claramente los componentes del problema, incluyendo:

- **Estado inicial**, es decir, la situación de partida o los datos de entrada disponibles.

- **Estado objetivo**, que refiere a la solución deseada o resultado final que se espera.

- **Condiciones y restricciones**, donde cualquier limitación o regla debe ser respetada por la solución.

- **Determinación de la naturaleza computacional.** Se evalúa si el problema puede ser resuelto paso a paso por una computadora. Algunos problemas pueden ser de decisión, con respuestas de sí o no, o bien de optimización, donde se busca la mejor solución.

Ejemplo de la fase de identificación de un problema

Considérese el problema de "encontrar la ruta más rápida para llegar a un destino".

1. **Análisis del problema:** un individuo necesita una forma eficiente de planificar un viaje. El problema no es solo llegar, sino hacerlo en el menor tiempo posible, evitando el tráfico o los retrasos.

2. **Definición de los elementos:**
   a. **Estado inicial:** la ubicación actual de la persona.
   b. **Estado objetivo:** la dirección del destino final.
   c. **Condiciones y restricciones:** la ruta debe considerar datos en tiempo real como el tráfico, la velocidad de las carreteras, posibles desvíos y el tipo de transporte.

3. **Identificación de la naturaleza computacional:** este es un problema ideal para una solución computacional, ya que se puede representar como un grafo con nodos (ubicaciones) y aristas (carreteras), y se puede usar un algoritmo para encontrar el camino más corto.

¿Sabías qué...?

En el mundo del emprendimiento y la tecnología, la fase de identificación no se llama simplemente "encontrar un problema", sino "definir una oportunidad de mercado". Los innovadores más exitosos no inventan cosas nuevas; simplemente identifican una fricción o un dolor que tiene la gente y luego usan el pensamiento computacional para crear una solución eficiente.

La fase de identificación es tan crítica que se compara a menudo con la Abstracción (cuarto pilar del pensamiento computacional), pero es necesario ignorar los síntomas para concentrarse en la causa de la raíz del problema. Por ejemplo, si un estudiante dice: mi problema es que no tengo suficiente tiempo para estudiar; la solución obvia sería que programara un horario de estudio, pero esta solución podría fallar, porque el verdadero problema podría ser: me distraigo fácilmente y no priorizo tareas. De ahí que la identificación incorrecta de un problema es la principal razón por que proyectos tecnológicos complejos y muy costosos fracasan. Los programadores resuelven perfectamente lo que se les pide, pero si la petición original no abordaba la verdadera necesidad, el resultado final es inútil.

Por tanto, no identificar el problema correcto es programar una solución equivocada.

Recurso digital

Escanea el QR para descargar el archivo del problema Piso *gamer* del salón de baile.

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza la siguiente actividad: **Preguntas de identificación del problema.**

1. Descarga el archivo del problema Piso *gamer* del salón de baile escaneando el código QR de al lado.

2. Analiza el problema y responde las siguientes preguntas que ayudan a la identificación del problema:

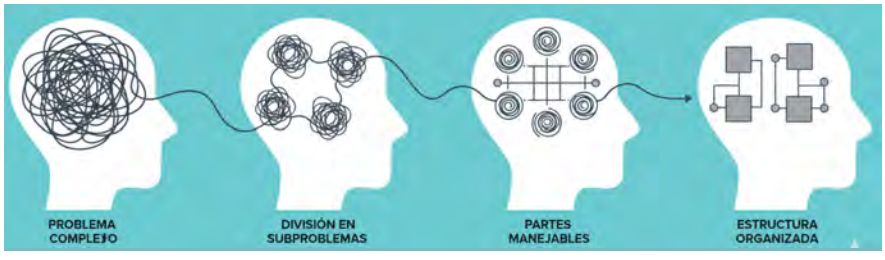
- **a.** ¿Cuál es el objetivo del problema?
- **b.** ¿Qué datos conoces desde el inicio?
- **c.** ¿Qué información falta por descubrir?
- **d.** ¿Cuál será el resultado que se quiere obtener?

3. Guarda tus respuestas en un documento de *Word* usando en el nombre tus iniciales seguidas de **_PC_P1_E01**.

4. Hazlo llegar a tu profesor por el medio que acuerden para que evalúe tus respuestas.

Descomposición del problema

Una vez que se ha completado la fase de identificación se procede a la siguiente etapa, la **descomposición**, que consiste en dividir el problema complejo o un sistema grande en partes más pequeñas, manejables e independientes, es decir, en subproblemas o módulos más sencillos de resolver de forma individual. En una situación sencilla como hornear un pastel, los subproblemas son preparar la masa, hornear y decorar.



Se puede decir que identificar el problema es el paso inicial para la descomposición.

Se recomienda llevar a cabo esta fase planteándose preguntas como:

- ¿Qué subproblemas o tareas lo componen?
- ¿Qué se necesita para resolver cada uno de ellos?

Ejemplo de la fase de descomposición de un problema

A la gente le cuesta mucho encontrar un taxi libre en la calle.

Descomposición:

- Subproblema 1:

¿Cómo saber dónde están los taxis? requiere GPS.

- Subproblema 2:

¿Cómo comunicar al conductor que necesito uno? requiere una *app* de solicitud.

- Subproblema 3:

¿Cómo pagar de forma segura? requiere un sistema de pagos integrado.

El Resultado:

Servicios como *Uber* o *DiDi* que nacen de la correcta identificación y descomposición de la frustración de buscar un taxi, convirtiéndolo en un algoritmo de conexión eficiente.

Ejercitando mis conocimientos

Para reforzar tu aprendizaje realiza de manera individual y con la guía de tu profesor la siguiente actividad:

Tabla de descomposición de problemas.

1. Aplica la fase Descomposición al problema *Piso gamer* del salón de baile.

2. Abre el documento de *Word* de Identificación del problema de la actividad anterior e inserta una tabla con el siguiente formato y responde en las celdas las respuestas necesarias a cada pregunta.

Subproblema	Descripción	¿Qué se necesita para resolverlo?	Posible resultado

3. Guarda el archivo usando en el nombre tus iniciales seguidas de **_PC_P1_E02** y compártela con tu profesor por el medio que acuerden.

¿Sabías qué...?

La necesidad de descomponer problemas complejos se popularizó durante la Segunda Guerra Mundial, no en la programación, sino en la ingeniería de sistemas. Proyectos enormes como el diseño de submarinos eran imposibles de manejar por una sola persona o un solo equipo. Los ingenieros dividieron el proyecto en sistemas más pequeños e interconectados. Esto dio origen al concepto de "modularidad" demostrando que cualquier meta gigante, desde construir un cohete hasta diseñar una *app*, solo se logra si se divide en módulos que funcionan de manera independiente pero armónica.

Reconocimiento de patrones

Es un paso clave para pasar de la comprensión del problema a la creación de una solución automatizada y generalizable.

El **reconocimiento de patrones** se hace identificando tendencias, estructuras, conexiones, semejanzas o regularidades entre las partes, esto es, en los datos o problemas, para resolverlos de manera más eficiente. Su importancia radica en que permite reutilizar soluciones pasadas o aplicar una técnica probada a varios subproblemas y no tener que reinventar la rueda cada vez que se enfrentan a una nueva situación. Lo que representa un enorme ahorro de tiempo y esfuerzo.

Al encontrar patrones, se pueden simplificar problemas complejos, crear soluciones repetibles y generalizar la resolución de problemas similares, por ejemplo, reconocer que todos los gatos tienen cola, ojos y pelaje permite dibujar un gato básico y luego añadir detalles específicos, en lugar de tener que definir cada gato desde cero.

El reconocimiento de patrones puede hacerse con esta guía:

1. **Identificación de similitudes:** usando la habilidad de ver lo que es igual o repetitivo en diferentes situaciones.

2. **Simplificación de problemas:** identificando patrones entre los problemas pequeños en los que se ha descompuesto uno complejo, lo que facilita su comprensión y resolución.

3. **Resolución eficiente:** creando soluciones repetibles para problemas similares, como usar un bucle para repetir una acción en programación en lugar de escribir el mismo código una y otra vez.

4. **Generalización:** haciendo predicciones y generalizando soluciones a partir de un conjunto de datos.

Ejemplo de la fase de reconocimiento de patrones

El docente escribe en el pizarrón las siguientes secuencias:

1. 2, 4, 8, 16, 32, …
2. 1, 1, 2, 3, 5, 8, 13, …
3. A, C, F, J, O, …

Los estudiantes deben encontrar la regla o patrón que explica cómo se genera cada secuencia.

Preguntas guía:
Discuten cómo reconocer una regularidad para predecir el siguiente valor y generar un algoritmo. Se apoyan respondiendo:

- ¿Qué relación hay entre un número (o letra) y el siguiente?
- ¿Se repite algún tipo de operación o salto?
- ¿Podría expresarse el patrón con una fórmula o con instrucciones?

¿Sabías qué…?

El sistema de las redes sociales analiza patrones de comportamiento de millones de usuarios simultáneamente. Sí el 80% de las personas que vieron el video A también interactuaron con el video B, hay un patrón. Así que cuando una canción se vuelve viral en plataformas como *TikTok* o un video es tendencia en *YouTube*, no es casualidad. El corazón de estas plataformas es un algoritmo de reconocimiento de patrones que predice tus próximos deseos.

Reconocimiento de patrones:
×2, suma de los dos anteriores y aumento progresivo de posiciones en el alfabeto.

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza la **Tabla de reconocimiento de patrones**:

1. Retoma el problema de *Piso gamer* del salón de baile.

2. Con base en la tabla de descomposición del problema, enfócate en encontrar patrones básate en las siguientes preguntas:

- **a.** ¿Hay simetría?
- **b.** ¿Las diagonales siguen un mismo patrón?
- **c.** ¿El diseño se repite igual si giras el cuadrado?

3. Encuentra los patrones que tiene el diseño del piso, por ejemplo puedes dividir el piso en cuatro partes o comparar lo que observas en cada una.

4. Inserta en el archivo una tabla con las siguientes columnas y rellénala con la información que observes en los patrones:

- **a.** Partes comparadas
- **b.** Similitudes
- **c.** Diferencias
- **d.** Posible patrón

5. Guarda el archivo usando en el nombre tus iniciales seguidas de **_PC_P1_E03** y compártela con tu profesor para que la evalúe.

Relaciónalo con...

Un error común que los estudiantes cometen al realizar la fase de abstracción es confundir representación con algoritmos, pero, elegir una buena abstracción no es lo mismo que implementar el algoritmo, ambas etapas deben aparecer por separado.

Abstracción

La **Abstracción** es el proceso de seleccionar y preservar sólo la información relevante de un problema y omitir los detalles que no afectan la solución, es el arte de la simplificación, filtrar la realidad para quedarse con lo esencial. Esta selección permite modelar el problema a un nivel manejable y útil para diseñar un algoritmo o sistema. Además, un buen modelo abstracto facilita el análisis y generalización a problemas similares, reduciendo la complejidad y guiando la implementación. Por el contrario, una abstracción pobre lleva a soluciones rígidas y erróneas.

La abstracción ha sido clave en la evolución del pensamiento computacional y sus aplicaciones. Es el puente entre el mundo real y su representación computacional.

Proceso práctico de la fase de abstracción es:

1. **Definir el objetivo:** tener claro lo que se quiere resolver.
2. **Listar los detalles del mundo real:** recopilar toda la información disponible.
3. **Preguntarse por la relevancia:** determinar la información esencial requerida para la solución, puede ser preguntándose ¿esta información influye en la solución? Omitir el ruido o los detalles que no afectan el resultado final.
4. **Elegir representaciones:** crear un modelo simplificado del problema, esto puede ser en una lista, matriz, grafo, conjunto de atributos, etc.
5. **Formalizar operaciones:** definir qué acciones, consultas o herramientas se necesitan sobre la representación.
6. **Probar la abstracción:** aplicar la representación a ejemplos concretos; verificar si permite resolver el objetivo.
7. **Iterar:** ajustar la abstracción, es decir, añadir o quitar atributos, según pruebas y errores.

Ejemplo de la fase de abstracción

Diseñar una plataforma que detecte automáticamente publicaciones que anuncian eventos escolares como talleres, torneos, conferencias, para agruparlas en un calendario.

Objetivo:
Crear un modelo que identifique si una publicación es anuncio de evento o no lo es.

Paso 1.
Definir el objetivo: clasificar cada publicación como "evento" o "no evento" con precisión razonable.

Paso 2.
Listar detalles: texto completo de la publicación (oraciones, *emojis*, *hashtags*), autor (perfil), fecha de publicación, imágenes adjuntas, enlaces externos, comentarios y reacciones.

Paso 3.
Decidir relevancia:

- Conservar (probablemente sean relevantes) presencia de palabras clave como "taller", "conferencia", "inscripciones", "fecha", "hora", "sede"; formato con fecha/hora, *hashtags* relacionados, enlaces a formularios de inscripción.
- Omitir en primera versión, las imágenes (porque requieren visión por computadora), tono emotivo (a menos que se use NLP avanzado), gran cantidad de comentarios (ruido).
- Consideración adicional: autor verificado puede aumentar probabilidad (atributo opcional).

Paso 4.
Elegir representación: representamos cada publicación como un vector de características binarias: 1= Evento, 0 = No evento

- f1: contiene palabra "taller" (0/1)
- f2: contiene palabra "inscripción" (0/1)
- f3: contiene una fecha (0/1)
- f4: contiene hora (0/1)
- f5: contiene hashtag relacionado (0/1)
- f6: longitud de texto (número)

Este vector es una abstracción: de texto largo y pocos atributos relevantes.

Paso 5.
Formalizar operaciones:

- Clasificar como evento mediante un árbol, usando los vectores relevantes etiquetados.

```
Si f3 = 1 (contiene fecha o día)
    ├─ Si f1 = 1 → Evento = 1
    ├─ Si f1 = 0
    │      ├─ Si f2 = 1 → Evento = 1
    │      ├─ Si f2 = 0
    │           ├─ Si f5 = 1 → Evento = 1
    │           └─ Si f5 = 0 → Evento = 0
Si f3 = 0 (sin fecha)
    ├─ Si f1 = 1 → Evento = 1
    └─ Si f1 = 0 → Evento = 0
```

Paso 6.
Probar la abstracción con los siguientes 3 ejemplos:

1. "¡Inscripciones abiertas para el taller de robótica este viernes a las 10:00! Regístrate aquí: …" ✅
2. "Miren este meme sobre los exámenes 😅" ✅
3. "Concurso de fotografía — más detalles mañana" ❌

Se detecta un falso negativo en ejemplo 3, la abstracción es rígida porque depende de palabras clave y de la presencia explícita de fecha/hora.

Paso 7.
Iterar y mejorar la abstracción:

- Añadir f7: contiene palabra "concurso" mejora la detección.
- Añadir detección de frases temporales: "mañana" y "este sábado" como f8.
- Considerar una puntuación y un umbral para clasificar.
- Probar con más ejemplos y ajustar.

Conceptos clave

NPL. Procesamiento de Lenguaje Natural (Natural Language Processing) es una rama de la IA que va más allá de la búsqueda de palabras clave, permite que una computadora comprenda el significado, la intención o el tono de los textos.

Relaciónalo con...

Un árbol de decisión es una estructura de tipo diagrama que organiza decisiones en ramas a partir de preguntas o condiciones. Cada nodo representa una pregunta, cada rama una respuesta posible (Si/No) y las hojas finales simbolizan una decisión o resultado. Modelar un árbol facilita ignorar detalles innecesarios, además se identifican las características clave al transformar información compleja en una estructura visual.

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza la **Tabla de abstracción**:

1. Retoma el archivo con las preguntas de identificación y las tablas de descomposición y descubrimiento de patrones que has trabajado las tres actividades anteriores con el problema Piso *gamer* del salón de baile.

2. Revisa las respuestas que tienes en el archivo y reflexiona:

a. ¿Qué partes del problema son realmente necesarias para explicar el diseño?
b. ¿Qué detalles podrías eliminar sin afectar su comprensión?
c. ¿Qué ideas o patrones se repiten y pueden generalizarse?

3. Inserta una tabla más en el mismo archivo, después de la tabla de descubrimiento de patrones, con las siguientes columnas:

a. Elemento del diseño o dato del problema.
b. ¿Es esencial?
c. Razón.
d Cómo puede representarse de forma simple.

4. Con base en las respuestas de la tabla elabora una representación gráfica en tu cuaderno de notas del diseño que conserve solo los elementos esenciales.

5. Escanea o toma foto con tu celular del esquema visual e insértalo debajo de la tabla de abstracción.

6. Guarda el archivo usando en el nombre tus iniciales seguidas de **_PC_P1_E04** y hazla llegar a tu profesor para recibir evaluación.

Diseño de algoritmo

Una vez que se ha analizado, descompuesto y abstraído el problema, el paso final antes de la implementación es crear la secuencia precisa y ordenada de instrucciones, es decir, el algoritmo.

Los algoritmos son la base de la programación de computadoras, son escritos en código especial entendible por los programas de computadora llamado lenguaje algorítmico. Este implementa una solución teórica a un problema indicando las operaciones a realizar y el orden en que deben ejecutarse.

Diseñar el algoritmo es traducir la solución abstracta en una secuencia de pasos claros, que puede ser en pseudocódigo, diagrama de flujo o instrucciones en lenguaje natural, que lleven a la resolución del problema. Su importancia es que traduce la comprensión abstracta del problema en un procedimiento ejecutable.

Algunas buenas prácticas en el diseño algorítmico son:

¿Sabías qué...?

No todos los algoritmos son iguales. Para un mismo problema, por ejemplo, ordenar una lista de números, pueden existir decenas de algoritmos. La clave es encontrar el algoritmo óptimo, la cual se mide por la eficiencia, que se divide en dos elementos: tiempo de ejecución y uso de memoria. Aspectos evaluados en concursos como la Olimpiada Mexicana de Informática.

Claridad	con pasos ordenados y sin ambigüedades.
Modularidad	dividir en funciones y/o subalgoritmos.
Condiciones y bucles	especificar cuándo repetir o tomar decisiones.
Entradas y salidas	definir claramente qué datos entran y qué resultados se esperan.
Casos límite	contemplar situaciones extremas, como datos faltantes o errores.

Ejemplo de diseño de algoritmo

Encender una lámpara solo si el interruptor está encendido.

Entrada:
- Estado del interruptor: puede ser ON o OFF.

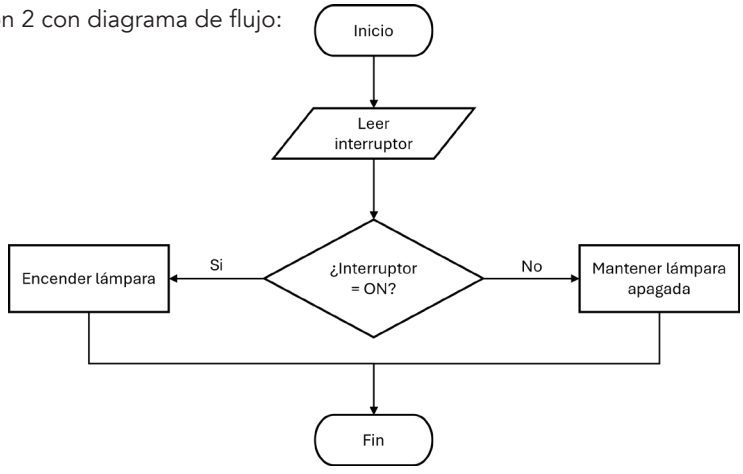
Proceso:
- Verificar el estado del interruptor.
- Si está ON, enviar corriente a la lámpara.
- Si esta OFF, mantener la lámpara apagada.

Salida:
- Estado de la lámpara, luz encendida o luz apagada.

Diseño del algoritmo:

Opción 1 con Pasos secuenciales en lenguaje natural:
1. Iniciar el estado del interruptor.
2. Leer el estado del interruptor.
3. Si el interruptor está en ON, encender la lámpara.
4. Si el interruptor está en OFF, dejar la lámpara apagada.
5. Termina el proceso.

Opción 2 con diagrama de flujo:



Relaciónalo con...

La fase de diseño es la parte del proceso que menos depende de una computadora. Los grandes diseñadores de algoritmos a menudo trabajan con lápiz y papel o una pizarra.

Conceptos clave

Lenguaje de programación estructurada. Es un tipo de lenguaje que organiza el código en una estructura lógica y modular, usando estructuras que controlan el flujo de la secuencia evitando saltos desordenados.

Opción 3 con pseudocódigo (en graphql):

```
Inicio
   Leer interruptor
   Si interruptor == ON entonces
       Encender lámpara
  Sino
       Mantener lámpara apagada
   Fin Si
Fin
```

Explicación paso a paso:

1. Inicio (se enciende o ejecuta el sistema).
2. Leer interruptor (el programa obtiene el valor actual del interruptor) 1= ON, 0=OFF).
3. Condición (el sistema evalúa si el interruptor está en ON).
4. Encender lámpara (si la condición se cumple, el programa activa la salida digital que alimenta la lámpara).
5. Sino (sino se cumple, la lámpara permanecerá apagada).
6. Fin (el programa concluye, sin repetir la secuencia).

Ejercitando mis conocimientos

Después de analizar el diseño del piso y descubrir sus patrones y estructura, es momento de convertir tus ideas en un algoritmo, es decir, una secuencia ordenada de pasos que cualquier persona pueda seguir para obtener el resultado correcto.

De manera individual y con la ayuda de tu profesor realiza un **Diseño de algoritmo** con las siguientes indicaciones:

1. Retoma el archivo generado la actividad anterior, el cual cuenta con la siguiente información:

- **a.** Preguntas de identificación
- **b.** Tabla de descomposición del problema
- **c.** Tabla de reconocimiento de patrones
- **d.** Tabla de abstracción del problema

2. Con base en toda la información redacta en el archivo un algoritmo que resuelva de manare eficiente el problema. Este debe cumplir con las características fundamentales de los algoritmos y atender las etapas de:

- **a.** Entrada de información
- **b.** Proceso de datos
- **c.** Salida de resultados

3. Una vez terminado guarda el archivo nombrándolo con tus iniciales seguidas de **_PC_P1_E05** y compártelo con tu profesor.

Implementación

La **implementación** es la traducción del algoritmo diseñado a un lenguaje que pueda ser ejecutado por una máquina. Significa pasar del algoritmo a una forma ejecutable, programarlo en un lenguaje o simular con herramientas visuales que permiten la automatización de la solución. Esta fase es el punto donde la solución teórica se convierte en un programa funcional.

A los usuarios que se inician en el campo de la algoritmia se les recomienda para ver sus algoritmos en acción emplear lenguajes de programación y entornos visuales de desarrollo sencillos como *Scratch*, *PSeInt*, *Code.org*, ya que ayudan a introducir conceptos sin la sintaxis propia de un lenguaje de programación estructurada como se requiere en *C*, *C++*, *Python*, *Java* y *Pascal*, entre muchos más.

Ejemplo de implementación de algoritmo

Un estudiante necesita saber si aprueba o reprueba una materia en función de su calificación final.

Entrada:
- Calificación (número del 0 al 10).

Proceso:
- Leer la calificación del estudiante.
- Evaluar si es mayor o igual a 6.
- Mostrar el resultado según corresponda.

Salida:
- Mensaje indicando si el estudiante aprueba o reprueba.

Algoritmo:
1. Iniciar el proceso.
2. Leer la *calificación* del estudiante.
3. Si la *calificación* es mayor o igual a 6, mostrar "Aprobado".
4. Si la *calificación* es menor que 6. Mostrar "Reprobado".
5. Terminar el proceso.

Pseudocódigo en PSeInt

```
1   Algoritmo Evaluar_calificación
2       Definir calificación Como Real
3
4       Escribir "Introduce tu calificación (0-10): "
5       Leer calificación
6
7       Si calificación ≥ 6 Entonces
8         Escribir "El estudiante está Aprobado. "
9       SiNo
10        Escribir "El estudiante está Reprobado. "
11      FinSi
12
13  FinAlgoritmo
```

¿Sabías qué...?

Aunque el lenguaje de programación Pascal es menos utilizado en proyectos modernos, es históricamente importante ya que fue diseñado para enseñar los principios de este tipo de programación.

Relaciónalo con...

Es conveniente que en el diseño de algoritmo y diagramas de flujo, se acostumbre a escribir las variables y constantes con formato de cursivas para identificarlas fácilmente.

Conceptos clave

Bugs. Es un error, fallo o defecto de sintaxis o de lógica dentro del código.

Bucle while. Es una estructura de control de flujo en programación que ejecuta repetidamente un bloque de código mientras una condición especificada es verdadera, se evalúa la condición antes de cada iteración y si es falsa, se termina el ciclo.

Bucle for. Estructura de control que se utiliza para ejecutar un bloque de código un número determinado de veces. Se usa comúnmente para iterar sobre colecciones de datos como listas o arreglos.

Evaluación

Esta última fase del pensamiento computacional, pero no menos importante, es la validación y mejora continua, aquí:

Se prueba → Se corrige → Se mejora

Evaluar significa medir si la solución cumple las metas definidas en la fase Identificación del problema. Involucra pruebas, análisis de eficiencia, de robustez frente a entradas no esperadas y de reflexión crítica sobre la solución y su impacto. Es decir que revela en el código del programa los errores de especificación o decisiones prácticas, de ahí que el resultado de la evaluación sirve de retroalimentación a la fase de implementación.

La evaluación implica principalmente tres acciones:

1. **Pruebas.** Esto es ejecutar el programa con diferentes datos de entrada, incluyendo casos limite, para verificar que se cumple con el objetivo.

2. **Depuración o *Debugging*.** Es el proceso de identificar, analizar y corregir en el código los errores, que a su vez se llaman bugs.

3. **Refinamiento.** Analizar la eficiencia y claridad de la solución. Aquí conviene hacerse preguntas como ¿funciona bien? ¿podría ser más rápido o consumir menos recursos?

Durante la realización de las pruebas se lleva a cabo la **iteración**, esto es, ejecutar un bloque de código una y otra vez. La repetición puede terminar una vez que se cumple una condición específica, por ejemplo un bucle *while* o cuando se ha ejecutado un número de veces definido, como un bucle *for*. Esta actividad es parte central del pensamiento computacional.

Es muy importante llevar a cabo la evaluación, pues asegura la fiabilidad y eficiencia del algoritmo, incluso las pruebas de escritorio son muy asertivas. Las pruebas simulan el comportamiento de un algoritmo, que se apoyan en una tabla con tantas columnas como variables tenga el algoritmo y seguir las instrucciones colocando los valores correspondientes.

Ejemplo de implementación de algoritmo

Comprobar que:
1. El algoritmo recibe correctamente la entrada (*calificacion*).
2. La condición lógica (>=6) se evalúa adecuadamente.
3. La salida corresponde al resultado esperado (Aprobado o Reprobado).
4. No hay errores lógicos ni omisión de casos límite.

Corrida de escritorio:

Caso	Entrada: calificación	Condición calificación ≥ 6	Resultado esperado	Salida generada por el algoritmo	Conclusión
1	9.5	Verdadero	Aprobado	El estudiante está aprobado	Correcto
2	6	Verdadero	Aprobado	El estudiante está aprobado	Correcto
3	5.9	Falso	Reprobado	El estudiante está reprobado	Correcto
4	0	Falso	Reprobado	El estudiante está reprobado	Correcto
5	10	Verdadero	Aprobado	El estudiante está aprobado	Correcto

Análisis de resultados:

- El algoritmo responde correctamente en todos los casos de prueba.
- Se evaluaron casos límite: 6 y 5.9, donde el comportamiento lógico fue el esperado.
- No hay errores en la estructura condicional ni en la salida de texto.
- Se cumple el objetivo: determinar si el estudiante aprueba o reprueba según su calificación.

El algoritmo cumple con la lógica del problema, produce resultados correctos en todos los escenarios de prueba, no requiere ajustes adicionales, por tanto, está listo para implementarse en un lenguaje de programación o entorno visual.

Ejercitando mis conocimientos

Realiza la actividad de **Implementación y evaluación**.

1. Retoma el archivo de la actividad anterior del algoritmo del problema del piso *gamer* salón de baile y comprueba el funcionamiento con una simulación de escritorio ejecutando el algoritmo paso a paso.

2. Realiza la implementación de tu algoritmo utilizando los datos de casos de prueba de la siguiente tabla:

Caso	Valor de N	Total de piezas	Piezas grises	Piezas blancas
1	23			
2	17			
3	9			
4	27			

Para saber más...

Accede al video Fases del pensamiento computacional, donde se explica la aplicación de cada una de las etapas con un problema específico. Accede a él escaneando el Código QR.

3. Llena la tabla con los valores obtenidos después de probar tu algoritmo con los valores de N de la tabla.

4. Pide a tu profesor que muestre en pizarrón su corrida y compara con tus resultados.

5. Una vez terminada la verificación, guarda el archivo usando en el nombre tus iniciales seguidas de **_PC_P1_E06** y compartelo con tu profesor.

Concretando mis conocimientos

Es momento de demostrar tu aprendizaje de las Fases del pensamiento computacional, para ellos de manera individual realiza la actividad y aplicar cada una de las etapas:

El propietario del *Pixel Market*, una tienda de videojuegos retro quiere digitalizar su sistema de caja.

Demuestra que eres un excelente programador y ayúdale a diseñar un algoritmo secuencial que le permita descomponer cualquier cantidad de dinero en monedas de distintas denominaciones: 500, 200, 100, 50, 25, 10, 5 y 1 pesos.

1. Crea un documento en *Word* para colocar el texto del problema y las tablas necesarias para aplicar las fases antes mencionadas.

a. Aplica la fase de identificación y descomposición del problema.
b. Observa que sucede y reconoce los patrones del problema.
c. Realiza la abstracción, conservado únicamente lo esencial del problema.
d. Diseña el algoritmo adecuado para resolver el reto.
e. Implementa y evalúa los resultados con la siguiente tabla de ejemplos:

Caso	Cantidad Total	Respuesta
1	385	hay 0 moneda(s) de 500 hay 1 moneda(s) de 200 hay 1 moneda(s) de 100 hay 1 moneda(s) de 50 hay 1 moneda(s) de 25 hay 1 moneda(s) de 10 hay 0 moneda(s) de 5 hay 0 moneda(s) de 1
2	987	

Caso	Cantidad Total	Respuesta
3	372	
4	624	
5	1568	

2. Llena la tabla anterior con los resultados de las pruebas de cada caso.

3. Una vez probado el algoritmo, guarda el documento usando en el nombre tus iniciales seguidas de **_PC_P1_CMC**. Hazlo llegar a tu profesor por el medio que acuerden para recibir evaluación.

Instrumento de evaluación

Revisa la siguiente lista de cotejo para que conozcas los criterios con los que tu profesor evaluará tu reporte escrito.

Indicador	Si	No	Puntos
Identificar las necesidades, restricciones y objetivos que debe cumplir el reto			1
Aplica la fase de descomposición			1
Reconoce los patrones del problema			3
Realiza la abstracción, conservado únicamente lo esencial del problema.			2
Diseña el algoritmo adecuado para resolver el reto			2
Implementa y evalúa los resultados con una corrida de escritorio según los valores de la tabla			1

Demostrando mi aprendizaje

Para demostrar tu aprendizaje conceptual referente a los temas abordados en la Progresión 1, realiza la actividad interactiva, ingresa a ella escaneando el código QR.

Progresión

2 Algoritmia en IDE

Resuelve problemas cotidianos y académicos mediante la construcción de algoritmos en IDE, utilizando estructuras de control decisivas e iterativas para automatizar procesos y validar la solución de manera digital.

Tiempo estimado: 12 horas

Tus metas serán:

- Identificar situaciones de la vida cotidiana que pueden resolverse de manera más eficiente utilizando secuencias y ciclos.
- Comprobar la lógica y funcionamiento de algoritmos para representar sus soluciones mediante IDE corrigiendo errores y optimizando el código.

Recuperando lo que sabemos

Este cuestionario es de recuperación de conocimientos previos, es útil para identificar tus saberes y habilidades y cómo los relacionas con la realidad, además te ayudará a comprender mejor los temas de esta secuencia. No es necesario que conozcas los términos técnicos; lo importante es expresar cómo entiendes o aplicarías cada situación, haz tu mejor esfuerzo y detecta aquellos aspectos que no conoces o dominas para enfocar tu estudio.

1. ¿Qué entiendes por algoritmo y qué papel crees que desempeña en la resolución de problemas dentro de la programación?

_____

_____

2. ¿Has utilizado antes algún programa o aplicación para escribir y ejecutar instrucciones o códigos? Si es así, ¿cuál fue tu experiencia?

_____

_____

_____

3. ¿Qué consideras que es un Entorno de Desarrollo Integrado (IDE) y cuál podría ser su función en la programación?

_____

_____

4. ¿Qué significa que un lenguaje de programación o pseudolenguaje tenga una sintaxis definida? ¿Por qué crees que es necesario respetarla?

_____

_____

_____

Reactivando mis conocimientos

Cada vez que organizas tu tiempo, divides una tarea en pasos o buscas la forma más rápida para lograr un objetivo, estás aplicando lógica, secuencia y análisis de problemas, los mismos principios que se utilizan al crear un algoritmo.

Imagina este escenario:

Después de la escuela, usas tu celular y te das cuenta de que tienes demasiadas notificaciones: mensajes, correos, avisos de redes y recordatorios. Tu objetivo es ordenar tus notificaciones para atender primero las más importantes (por ejemplo, una tarea, un mensaje urgente o una alerta del calendario). Para lograrlo, necesitas pensar en un procedimiento paso a paso que te ayude a decidir qué notificaciones revisar primero y cuáles después.

1. En tu cuaderno o documento digital, escribe los pasos que seguirías para organizar tus notificaciones de manera lógica y eficiente.

2. Identifica los elementos del problema:

   - Datos de entrada: ¿Qué información recibes? (mensajes, horarios, alertas, etc.)
   - Proceso: ¿Qué acciones o reglas aplicas para decidir el orden de atención?
   - Salida: ¿Cuál es el resultado final o estado ideal de tu pantalla?

3. Reflexiona y responde en tus notas:

   - ¿Qué parte de tu procedimiento crees que un programa podría automatizar?
   - ¿Cómo te ayudaría usar un entorno de desarrollo para simular tu algoritmo antes de programarlo?
   - ¿Qué ventajas tendría poder observar cómo se ejecutan tus pasos uno por uno en una simulación?

Comparte en clase tus pasos y reflexiones con tus compañeros y el profesor. Analicen juntos cuál de los procedimientos fue más claro, ordenado y eficiente, y comenten cómo ese mismo proceso podría transformarse en un algoritmo computacional.

2.1 Aplicación en entorno de desarrollo integrado

PSeint

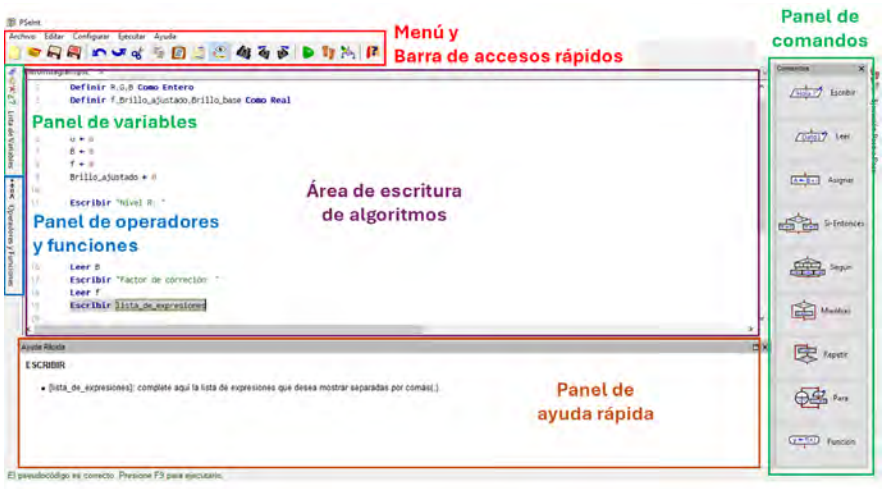
El desarrollo del pensamiento computacional requiere comprender cómo las ideas abstractas se transforman en soluciones concretas mediante algoritmos. En este proceso, los Entornos de Desarrollo Integrado (*IDE*, por sus siglas en inglés) desempeñan un papel esencial, ya que permiten diseñar, escribir, ejecutar y depurar programas dentro de una misma interfaz. El uso de un *IDE* favorece la práctica de la algoritmia al proporcionar herramientas que ayudan al programador a concentrarse en la lógica de solución más que en los aspectos técnicos del lenguaje.

Históricamente, los primeros entornos de programación consistían únicamente en editores de texto y compiladores separados. Con el avance de la ingeniería de *software*, surgió la necesidad de integrar todos los recursos en una sola plataforma. Así nacieron los *IDE*, que incorporan componentes como un editor de código, un compilador o intérprete, un depurador y, en algunos casos, simuladores o asistentes visuales. Esta evolución no solo mejoró la productividad del programador, sino que también simplificó el proceso de aprendizaje para quienes se inician en la programación.

En el contexto educativo, *PSeInt* (Intérprete de Pseudocódigo) representa una herramienta didáctica ideal para comprender los fundamentos de la algoritmia. Su diseño se orienta a facilitar la construcción de algoritmos utilizando un pseudolenguaje cercano al español, lo que reduce la complejidad sintáctica y permite enfocarse en la lógica del problema. *PSeInt* simula el funcionamiento de un lenguaje estructurado, favoreciendo la comprensión del flujo lógico y del proceso de ejecución de un programa.

Interfaz

La interfaz de *PSeInt* está diseñada para facilitar el trabajo con algoritmos, en ella se distinguen los siguientes elementos principales.



Interfaz de *PSeint*

Recurso digital

Escanea el código QR para descargar el archivo instalador de *PSeInt*.

- **Panel de Variables:** muestra las variables identificadas, organizadas por proceso y subprocesos. El ícono representa el tipo de dato.

- ***Panel de Operadores y Funciones:*** presenta un catálogo con las funciones y constantes predefinidas y la lista de posibles operadores, organizado por categorías. Al hacer clic sobre uno de ellos se inserta en el pseudocódigo.

- **Panel de Comandos:** permite introducir acciones o estructuras de control mediante un clic. Introduce el código del proceso seleccionado, marcando con recuadros las partes que se deben completar (expresiones, acciones, valores, etc.).

- **Panel de Ayuda Rápida:** brinda detalles y sugerencias para corregir los errores que el intérprete encuentre en el algoritmo, se despliega automáticamente en la parte inferior de la ventana cada vez que se introduce un comando mediante el Panel de Comandos o cada vez que se hace clic sobre un mensaje de error.

- **Panel de Ejecución Paso a Paso:** permite controlar de forma detallada la ejecución del algoritmo o configurar la prueba de escritorio, si no se encuentra visible se puede ejecutar al pulsar el comando ubicado en el margen derecho de la ventana o desde la barra de accesos rápidos. La prueba de escritorio consiste en realizar un seguimiento detallado de los valores que van tomando las variables en cada paso, *PSeInt* construye una tabla automáticamente mostrando las variables o expresiones seleccionadas.

Proceso/SubProceso	Linea(inst)	R	G	B	f	Brillo_base	Brillo_ajustado
1:FILTROINSTAGRAM	24(1)	13	56	5	0.6	43.4	26.04
1:FILTROINSTAGRAM	26(1)	13	56	5	0.6	43.4	26
1:FILTROINSTAGRAM	28(1)	13	56	5	0.6	43.4	26

Prueba de escritorio

Estudiando

Dedica un tiempo a la lectura de las páginas correspondientes al tema **Componentes del pseudolenguaje**. Realizar esta tarea te facilitará las actividades que el profesor guiará en las siguientes sesiones.

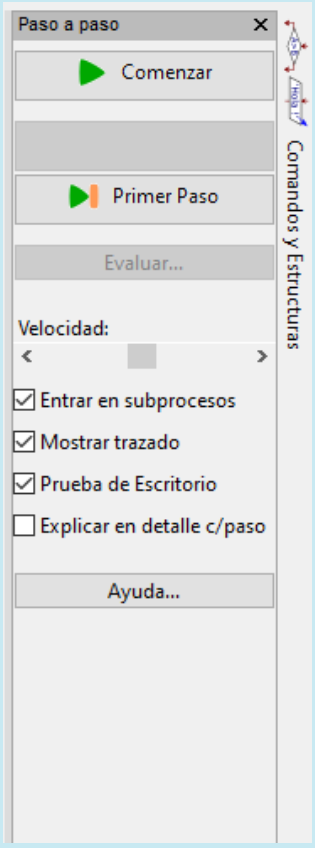
Componentes del pseudolenguaje

Todo algoritmo en pseudocódigo tiene la siguiente estructura general:

Comienza con la palabra clave **Algoritmo**, seguida del **nombre del programa**, luego le sigue una secuencia de **instrucciones** y finaliza con la palabra **FinAlgoritmo**. Una secuencia de instrucciones es una lista de una o más instrucciones y/o estructuras de control.

```
Algoritmo Titulo
    acción 1;
    acción 2;
        .
        .
    acción n;
FinAlgoritmo
```

Botón de función ejecutar paso a paso



Panel de ejecución paso a paso

Para saber más...

Escanea el código QR y observa el video Interfaz de *PSeInt*.

Conceptos clave

Palabras reservadas. Son términos con un significado específico en *PSeInt*, como *Algoritmo*, *Como*, *Leer* o *Escribir*. Forman parte del lenguaje del algoritmo y no pueden usarse como nombres de variables.

¿Sabías qué...?

En *PSeInt* existen constantes predefinidas como *PI* y *E* (número de Euler), que representan valores matemáticos universales. Estas pueden usarse directamente en los algoritmos sin necesidad de declararlas, facilitando cálculos con mayor precisión.

```
Dias_semana ← 7
```

Constante

Variables

Una **variable** representa un espacio de memoria destinado a almacenar información temporal durante la ejecución de un programa. Por ejemplo, si se desea calcular el área de un triángulo, es necesario guardar los valores de la base y la altura en variables, para luego realizar la operación correspondiente y almacenar el resultado en otra. El valor contenido en una variable puede modificarse conforme avanza la ejecución del programa, lo que le otorga su carácter dinámico. En pocas palabras, una variable es un contenedor que permite guardar y manipular datos.

Cada variable se identifica mediante un **nombre** o **identificador**, el cual debe seguir ciertas reglas para evitar ambigüedades. Un identificador válido inicia con una letra y puede incluir letras, números o guiones bajos, pero no admite espacios, operadores ni coincidencias con palabras reservadas del lenguaje. Algunos ejemplos podrían ser: *A, Lado1, Total, Nombre_Apellido* o *DireccionCorreo*. En la mayoría de los lenguajes de programación, los nombres de variables no pueden contener caracteres especiales (acentos, diéresis, letra "ñ").

En *PSeInt*, toda variable está asociada a un tipo de dato, lo que implica que solo puede almacenar valores del mismo tipo durante la ejecución. Por ejemplo, una variable declarada para guardar números no puede utilizarse posteriormente para almacenar texto.

Constantes

Una **constante** es un espacio de memoria cuyo valor permanece invariable durante toda la ejecución del algoritmo. A diferencia de las variables, las constantes se utilizan para representar datos fijos, como valores matemáticos o parámetros que no deben modificarse, garantizando así la estabilidad y legibilidad del programa.

El uso de constantes evita errores por cambios accidentales en datos esenciales y mejora la comprensión del algoritmo, ya que permite identificar fácilmente valores que poseen un significado específico dentro del programa.

Tipos de datos

Los tipos de datos determinan la clase de información que una variable puede almacenar y las operaciones que se pueden realizar con ella. En *PSeInt* los tipos básicos son *entero, real, carácter* y *lógico*.

Tipo	Descripción	Ejemplos de Valor
Entero	Representa números sin decimales. Se utiliza para contar o realizar operaciones aritméticas exactas.	15, -10, 0
Real	Almacena números con parte decimal. Permite realizar cálculos con precisión fraccionaria.	8.75, -2.5, 0.33
Carácter	Contiene un carácter o cadenas de caracteres encerrados entre comillas. Es útil para manejar datos de texto individuales, palabras o mensajes.	"A", "5", "?", "Carlos"
Lógico	Representa valores de verdad, empleados en condiciones o decisiones del algoritmo.	Verdadero, Falso

Operadores

En un algoritmo, los operadores son símbolos que permiten realizar operaciones entre valores o variables. Constituyen elementos esenciales del lenguaje, ya que posibilitan el procesamiento de datos y la toma de decisiones dentro del programa. En *PSeInt*, los operadores se agrupan según la función que desempeñan: aritmética, relacional y lógica.

Los **operadores aritméticos** se utilizan para efectuar cálculos numéricos con variables de tipo entero o real. Permiten realizar operaciones básicas como suma, resta, multiplicación, división o cálculo del residuo de una división y se emplean dentro de condiciones y estructuras de control.

Operadores Aritméticos	Función	Ejemplo	Resultado
+	Suma	5 + 3	8
-	Resta	10 - 4	6
*****	Multiplicación	2 * 3	6
/	División	9 / 3	3
% o MOD	Módulo (residuo)	10 % 3	1

Los **operadores relacionales** comparan dos valores y devuelven un resultado lógico: Verdadero o Falso. Son indispensables en estructuras condicionales, donde el flujo del algoritmo depende del cumplimiento de una condición.

Operadores Relacionales	Función	Ejemplo	Resultado
=	Igual a	A = B	Verdadero si A y B son iguales
<>	Distinto que	A <> B	Verdadero si A y B son diferentes
<	Menor que	A < B	Verdadero si A es menor que B
>	Mayor que	A > B	Verdadero si A es mayor que B
<=	Menor o igual que	A <= B	Verdadero si A es menor o igual que B
>=	Mayor o igual que	A >= B	Verdadero si A es mayor o igual que B

Los **operadores lógicos** se utilizan para combinar o modificar expresiones relacionales. Permiten evaluar condiciones complejas y controlar la lógica del programa, resultan esenciales para la toma de decisiones y la repetición controlada de procesos dentro de un algoritmo.

Operadores Lógicos	Función	Ejemplo	Resultado
& o Y	Conjunción lógica (AND)	(A > 0) & (B < 10)	Verdadero si ambas condiciones se cumplen
| u O	Disyunción lógica (OR)	(A = 5) | (B = 7)	Verdadero si al menos una condición se cumple
~ o NO	Negación lógica (NOT)	~ (A = 5)	Verdadero si A no es igual a 5

Para saber más...

PSeInt ofrece funciones predefinidas que permiten realizar cálculos, manejar texto o generar números aleatorios con solo una instrucción.

Escanea el QR para acceder a un documento con las funciones predefinidas más comunes.

¿Sabías qué...?

En *PSeInt* existen acciones secuenciales especiales. Por ejemplo, **Limpiar Pantalla**, **Esperar Tecla**, que detiene el algoritmo hasta presionar una tecla y **Esperar**, que pausa la ejecución durante un tiempo específico.

El dominio de los operadores en *PSeInt* permite traducir razonamientos matemáticos y lógicos en instrucciones comprensibles para el intérprete. Su uso adecuado garantiza la correcta evaluación de expresiones, la manipulación precisa de datos y la ejecución coherente de algoritmos dentro del entorno de desarrollo.

▶ Acciones primitivas secuenciales

Las acciones primitivas secuenciales representan las instrucciones más básicas y directas que un algoritmo puede ejecutar en *PSeInt*. Se denominan primitivas porque constituyen operaciones elementales, que no se pueden descomponer en pasos más simples dentro del pseudolenguaje. A su vez, se consideran secuenciales porque cada instrucción se ejecuta únicamente después de que la anterior haya finalizado, garantizando una secuencia ordenada y predecible.

Entre las principales acciones primitivas secuenciales en *PSeInt* se encuentran:

Acción	Descripción	Ejemplo
Definir	Declara una variable indicando su tipo de dato	Definir edad Como Entero
Asignar	Establece el valor de una variable	edad ←16
Leer	Permite ingresar información desde el teclado y almacenarla en una variable	Leer edad
Escribir	Muestra en pantalla mensajes o resultados almacenados en variables	Escribir "Tu edad es: ", edad

Estas acciones son fundamentales para crear programas sencillos y entender cómo se ejecutan paso a paso los algoritmos. Gracias a ellas, es posible ingresar datos, procesarlos y mostrar resultados sin utilizar condiciones ni repeticiones. Dominar las acciones primitivas secuenciales ayuda a comprender la lógica de la programación estructurada, ya que permite reforzar los conceptos de entrada, proceso y salida, base del pensamiento algorítmico.

Recurso digital

Escanea el código QR, descarga el documento y sigue las instrucciones de la actividad con la guía de tu profesor.

Ejercitando mis conocimientos

Para reforzar tu aprendizaje realiza de manera individual y con la guía de tu profesor la siguiente actividad:

1. Descarga el archivo de *MS Word* escaneando el código QR de esta página.

2. Aplica las fases del pensamiento computacional y utiliza el IDE PSeInt para diseñar, implementar y evaluar el algoritmo que resuelva el problema planteado en el documento que descargaste.

3. Una vez realizado el algoritmo en *PSeInt* activa la ejecución paso a paso y haz un seguimiento del proceso capturando la información o valores correspondientes en cada celda de la tabla de ejecución paso a paso en el archivo de *MS Word* que descargaste.

4. Guarda el algoritmo en *PSeInt* y el documento utilizando en el nombre de ambos archivos tus iniciales seguidas de **_PC_P2_E01**.

5. Comparte ambos archivos con tu profesor para recibir retroalimentación por el medio que acuerden.

2.2 Estructuras de control

Una estructura de control es un mecanismo que posibilita alterar el orden natural de ejecución de las instrucciones dentro de un programa en el ámbito de la programación.

Las instrucciones por defecto se ejecutan de manera secuencial, es decir, una tras otra. Sin embargo, en la mayoría de los algoritmos es necesario tomar decisiones, como cuando se ejecuta un bloque de código únicamente si se cumple una condición, o bien realizar tareas repetitivas, como calcular promedios de varios valores o recorrer una lista.

Por esta razón, se utilizan tres tipos fundamentales de estructuras de control:

1. **Secuenciales**, donde las instrucciones se ejecutan en orden lineal.
2. **Condicionales**, donde el flujo depende de una o varias condiciones lógicas.
3. **Repetitivas**, donde un conjunto de instrucciones se ejecuta varias veces.

Además de facilitar la toma de decisiones y la repetición de tareas, las estructuras de control permiten escribir programas más eficientes, legibles y fáciles de mantener. Gracias a ellas, los desarrolladores pueden dividir problemas complejos en bloques lógicos más simples, lo que mejora la organización del código y reduce la posibilidad de errores.

En entornos educativos como *PSeInt*, estas estructuras son fundamentales para que los estudiantes comprendan cómo fluye la lógica en un algoritmo y cómo se puede controlar ese flujo para resolver problemas de manera efectiva.

¿Sabías qué...?

Un simple error en el orden secuencial de las instrucciones puede cambiar completamente el resultado de un programa.

Estructura secuencial

Este tipo de estructura es la más simple y también la más fácil de aplicar. En ella, las instrucciones se ejecutan una detrás de otra, sin saltos ni repeticiones, siguiendo únicamente el orden en que han sido escritas.

Estructura secuencial

Este tipo de estructura es ideal cuando todas las acciones deben realizarse exactamente una a la vez y en el mismo orden.

Con el fin de ilustrar de manera más precisa el uso de la estructura **secuencial**, se propone la resolución del siguiente problema.

▶ *Combo Gamer RGB*

Luis, un joven streamer, está armando su *setup gamer* para transmitir sus partidas en *Twitch*. En su carrito tiene un *mouse gamer RGB*, unos audífonos con micrófono profesional y un tapete luminoso para el mouse.

Cada artículo tiene su precio, pero al pagar deberá sumarse el IVA del 16 %. Tu misión es ayudar a Luis a crear un algoritmo para calcular el total final de su compra.

¿Sabías qué...?

Usar correctamente los condicionales puede mejorar la eficiencia del programa al evitar operaciones innecesarias.

Entrada
Tres números reales separados por un espacio: el precio del mouse, de los audífonos y del tapete *RGB*.

Salida
Un número real con dos decimales: el total a pagar incluyendo IVA.

Entrada	Salida
399.90 649.50 199.00	1448.14

Algoritmo en *PSeInt*

```
Algoritmo ComboRGB
   Definir mouse,audifonos,tapete,total,iva Como Real
   mouse <- 0
   audifonos <- 0
   tapete <- 0
   total <- 0
   iva <- 0

   Escribir "Precio del mouse: "
   Leer mouse

   Escribir "Precio de los audífonos: "
   Leer audifonos

   Escribir "Precio del tapete: "
   Leer tapete

   total <- (mouse + audífonos + tapete) * 1.16
   total <- trunc(total*100)/100

   Escribir "Total a pagar: ",total

FinAlgoritmo
```

Estructuras condicionales

Este tipo de estructuras permiten ***"tomar decisiones"*** dentro de un algoritmo. En la vida cotidiana, se usan constantemente frases como:

- "Si llueve, llevaré paraguas."
- "Si es de noche, encenderé la luz; si no, la apagaré."
- "Si saco buena calificación en Pensamiento Computacional, celebraré; si no, estudiaré más."

En programación ocurre algo similar, el algoritmo ejecuta una acción **solo si** se cumple cierta condición lógica.

PSeInt utiliza la palabra clave **SI** para expresar este tipo de decisiones. Dependiendo de la complejidad, existen varias formas de estructuras condicionales: simple, doble, anidada y según la opción.

Condicional Simple

La estructura simple evalúa una condición lógica y si se cumple, ejecuta una o más instrucciones; de lo contrario, continúa con el flujo normal del programa.

La sintaxis de un condicional simple sería así:

```
Si (condición) Entonces
   // instrucciones a ejecutar si se cumple la condición
FinSi
```

El ejemplo que se muestra a continuación evidencia la aplicación del condicional simple.

Acceso al concierto

La banda de rock Jayler está por comenzar su concierto en la ciudad de Mazatlán, la entrada al concierto solo está permitida para mayores de 15 años.

Te han contratado para que ayudes a la banda a controlar el acceso a través de una aplicación, para eso necesitas crear un algoritmo que reciba la edad de una persona y determine si puede acceder al concierto.

Entrada
Un número entero que representara la edad del asistente.

Salida
Un mensaje con el texto "Acceso permitido. Disfruta el concierto!" si el asistente cumple con la condición para acceder al concierto.

Entrada	Salida
¿Cuál es tu edad? 16	Acceso permitido. Disfruta el concierto!
¿Cuál es tu edad? 15	

Algoritmo en *PSeInt*

```
Algoritmo ConciertoJayler
 Definir edad Como Entero
 edad <- 0
 Escribir "¿Cuál es tu edad?"
 Leer edad

 Si (edad > 15) Entonces
   Escribir "Acceso permitido. Disfruta el concierto!"
 FinSi

FinAlgoritmo
```

Estructura **condicional simple**

Para saber más...

Accede al video Estructura de control Simple en *PSeInt*, para ampliar la explicación del tema. Hazlo escaneando el Código QR.

¿Sabías qué...?

Puedes usar condicionales para mostrar diferentes mensajes, validar datos o controlar el flujo del programa.

Estructura **condicional doble**

Condicional doble

La estructura condicional doble agrega una alternativa cuando la condición no se cumple. En este caso, se usa la palabra clave *Sino*, que indica que instrucciones ejecutar en el caso de que no sea cumplida la condición.

La sintaxis de un condicional doble, sería así:

```
Si (condición) Entonces
   // Bloque de instrucciones si se cumple la condición
Sino
   // Bloque de instrucciones si NO se cumple la condición
FinSi
```

A continuación, se presenta un ejemplo que pone en práctica la estructura condicional doble.

Descuento Estudiantil GeekBooks

La tienda digital *GeekBooks* premia a los estudiantes aplicando un 10% de descuento en cualquier libro digital si presentan su credencial escolar.

Entrada
Un número real que indica el precio del libro,
Un carácter que indica si el cliente presenta credencial de estudiante o no (S=si, N=no)

Salida
Un mensaje compuesto por el texto: "Total a pagar: " unido con el número real que corresponde al monto total a pagar.

Entrada	Salida
¿Cuál es precio del libro digital? 250.00 ¿Credencial de estudiante? S	Total a pagar: 225.00
¿Cuál es precio del libro digital? 250.00 ¿Credencial de estudiante? N	Total a pagar: 250.00

Para saber más...

Escanea el código QR y observa el video Estructura condicional doble en *PSeInt*.

Algoritmo en *PSeInt*

```
Algoritmo DescuetoEstudiantil
Definir precio,total Como Real
Definir credencial Como Caracter

precio <- 0
total <- 0
credencial <- ''

Escribir "¿Cuál es el precio del libro?"
Leer precio
Escribir "¿Tienes credencial de estudiante? (S o N)"
Leer credencial

Si (credencial= 'S' O credencial= 's') Entonces
   total <- precio * .90
SiNo
   total <- precio
FinSi

total <- trunc(total * 100 ) / 100

Escribir "Total a pagar: ", total
FinAlgoritmo
```

Ejercitando mis conocimientos

Para reforzar tu aprendizaje realiza de manera individual y con la guía de tu profesor la siguiente actividad:

1. Descarga el archivo *PDF* escaneando el código QR donde encontraras un problema para aplicar el uso de la **estructura condicional simple y doble**.

2. Genera el algoritmo en *PSeInt*, y una vez terminado, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el algoritmo creado en *PSeInt* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P2_E02**.

4. Comparte ambos archivos con tu profesor para recibir retroalimentación por el medio que acuerden.

Recurso digital

Escanea el código QR para descargar el archivo del problema de estructura condicional simple y doble.

Estructura **condicional anidada**

Condicional anidada

En el desarrollo de algoritmos, una sola condición no siempre resulta suficiente para decidir qué debe hacer un programa. Cuando la decisión depende de varios factores al mismo tiempo, es necesario utilizar una estructura que permita revisar más de una condición de manera ordenada. En estos casos, las condicionales anidadas se convierten en un recurso importante, ya que permiten colocar una condición dentro de otra para establecer un orden lógico en la toma de decisiones.

El uso de **condicionales anidadas** responde a la necesidad de relacionar distintos criterios que influyen en el funcionamiento del programa. En muchos problemas, la primera condición solo sirve para identificar un conjunto general de posibilidades, y cada una de ellas requiere luego una verificación más específica. Esta forma de organización ayuda a que el algoritmo sea más preciso y pueda adaptarse a situaciones diversas, evitando soluciones demasiado simples o poco flexibles.

En la práctica, estas estructuras son especialmente útiles en algoritmos que deben realizar varias verificaciones, como sistemas de acceso seguro, cálculos con diferentes rangos de valores o programas que dependen de ciertos parámetros del entorno. En todos estos casos, las condicionales anidadas ayudan a que el programa responda de manera adecuada a distintas situaciones y evite errores lógicos.

Finalmente, aunque las condicionales anidadas son una herramienta valiosa, no siempre representan la mejor opción. Cuando el número de condiciones es muy grande o su relación es compleja, puede ser más conveniente utilizar otras estrategias, como tablas de decisión, operadores lógicos combinados o métodos propios de la programación orientada a objetos. Elegir la alternativa adecuada permite mejorar tanto el rendimiento del programa como la facilidad para entender y mantener el código.

El lenguaje *PSeInt* admite esta modalidad mediante la reiteración de bloques del tipo **Si…Entonces…Sino** dentro de otros similares, facilitando así una lógica de decisión más compleja y estructurada.

La sintaxis de una condicional anidada sería:

```
Si (condición1) Entonces
   // Bloque de instrucciones de condición1
Sino
   Si (condición2) Entonces
      // Bloque de instrucciones de condición2
   Sino
      // Bloque de instrucciones si no cumplen condición 1 y 2
   FinSi
```

Para saber más…

Accede al video Estructura de control Anidada en *PSeInt*, para ampliar la explicación del tema. Hazlo escaneando el Código QR.

El ejemplo siguiente tiene como propósito mostrar la estructura de **condicional anidada.**

▶ *Héroe digital*

En el videojuego "Héroes del Código", los jugadores obtienen recompensas diarias según su nivel y si completaron el reto del día.

Dependiendo de su esfuerzo, pueden recibir desde simples monedas hasta una *skin* épica legendaria.

Reglas del juego:

Nivel ≥ 20
Reto completado → Skin Épica
No completado → Skin Rara
Nivel < 20
Reto completado → Caja de Ítems
No completado → Monedas x100

Crea un algoritmo que sea capaz de determinar la recompensa del jugador.

Entrada
Un número entero que indica el nivel del jugador
Un carácter simboliza si se cumplió o no el reto (S=si, N=no)

Salida
Mensaje con la recompensa obtenida por el jugador

Entrada	Salida
¿Cuál es nivel del héroe? 22 ¿Reto completado? S	Skin Épica
¿Cuál es nivel del héroe? 22 ¿Reto completado? N	Skin Rara
¿Cuál es nivel del héroe? 19 ¿Reto completado? S	Caja de Items
¿Cuál es nivel del héroe? 19 ¿Reto completado? N	Monedas x100

Algoritmo en *PSeInt*

```
Algoritmo HeroeDigital
        Definir nivel Como Entero
        Definir reto Como Caracter

        nivel <- 0
        reto <- ‘’

        Escribir “¿Cuál es el nivel del jugador?”
        Leer nivel
        Escribir “¿Completó el reto diario? (S=si, N=no)”
        Leer reto

        Si (nivel >= 20 ) Entonces
               Si (reto= ‘s’ 0 reto= ‘S’) Entonces
                      Escribir “Skin Épica”
               SiNo
                      Escribir “Skin Rara”
               FinSi
        SiNo
               Si (reto= ‘S’ 0 reto= ‘s’) Entonces
                      Escribir “Caja de Ítems”
               SiNo
                      Escribir “Monedas x100”
               FinSi
        FinSi
 FinAlgoritmo
```

Recurso digital

Escanea el código QR para descargar el archivo del problema de la estructura condicional anidada.

Ejercitando mis conocimientos

Fortalece tu dominio de las estructuras condicionales anidadas elaborando de manera individual y con la ayuda de tu profesor la actividad propuesta:

1. Descarga el archivo PDF escaneando el código QR donde encontraras un problema para aplicar el uso de la de la **Estructura Condicional Anidada**.

2. Genera el algoritmo en *PSeInt*, y una vez terminado, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el algoritmo creado en *PSeInt* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P2_E03**.

4. Comparte ambos archivos con tu profesor para recibir retroalimentación por el medio que acuerden.

Estructura Según...Hacer

La estructura **Según...Hacer**, equivalente a *switch* o *case* en otros lenguajes de programación, se utiliza cuando es necesario tomar decisiones basadas en el valor de una sola variable con múltiples opciones posibles. Su uso resulta más ordenado y legible que el de varias estructuras condicionales anidadas, especialmente cuando existen muchas alternativas de ejecución.

La sintaxis de la estructura **Según...Hacer** es la siguiente:

```
Según variable Hacer
   Opción 1:
      // Instrucciones si variable = Opción 1
   Opción 2:
      // Instrucciones si variable = Opción 2
   De Otro Modo:
      // Instrucciones si no coincide ninguna opción
FinSegún
```

En el siguiente ejemplo se expone la aplicación de la estructura **Según...Hacer**.

Reacciones en Red Social

Una nueva red social llamada *MoodWave* registra las reacciones de los usuarios en publicaciones.

Cada reacción tiene un código numérico:

1. "Me gusta"
2. "Me encanta"
3. "Me asombra"
4. "Me divierte"
5. "Me entristece"

De otro modo: "Reacción no válida"

Esta red social esta reclutando gente para desarrollar el algoritmo que muestre la reacción correspondiente, así que diseña el algoritmo para quedarte con el puesto de programador.

Entrada
Un número entero que representa el código de la reacción

Salida
El mensaje que indica la reacción correspondiente al código, en caso de recibir un código que no existe en la tabla se debe mostrar el mensaje "Reacción no válida".

Entrada	Salida
¿Qué código de reacción deseas usar? (1-5): 4	Me divierte
¿Qué código de reacción deseas usar? (1-5): 1	Me gusta
¿Qué código de reacción deseas usar? (1-5): 6	Reacción no válida

Estructura repetitiva
según...hacer

Para saber más...

Escanea el código QR y observa el video Estructura condicional Según la opción en *PSeInt*.

Algoritmo en *PSeInt*

```
Algoritmo RedSocial
   Definir reaccion Como Entero
   reaccion <- 0

   Escribir "¿Qué código de reacción deseas usar? (1-5)"
   Leer reaccion

   Segun reaccion Hacer
       1: Escribir "Me gusta"
       2: Escribir "Me encanta"
       3: Escribir "Me asombra"
       4: Escribir "Me divierte"
       5: Escribir "Me entristece"
       De Otro Modo: Escribir "Reacción no válida"
   FinSegun
FinAlgoritmo
```

Recurso digital

Escanea el código QR para descargar el archivo del problema de la estructura condicional segun…hacer.

Ejercitando mis conocimientos

Para reforzar tu aprendizaje realiza de manera individual y con la guía de tu profesor la siguiente actividad:

1. Descarga el archivo *PDF* escaneando el código QR donde encontraras un problema donde aplicaras el uso de la **Estructura Condicional Segun…Hacer**.

2. Genera el algoritmo en *PSeInt*, y una vez terminado, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el algoritmo creado en *PSeInt* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P2_E04**.

4. Comparte ambos archivos con tu profesor para recibir retroalimentación por el medio que acuerden.

Estructuras repetitivas

En el desarrollo de algoritmos, es común encontrar situaciones en las que ciertas operaciones deben repetirse múltiples veces.

Escribir el pseudocódigo de estas instrucciones de forma repetitiva resulta poco práctico y aumenta la posibilidad de cometer errores. Por esta razón, se incorporan estructuras de control conocidas como **bucles, ciclos o estructuras repetitivas**, las cuales permiten ejecutar un conjunto de instrucciones de manera automatizada y controlada, optimizando la eficiencia del código y facilitando su compresión.

En el desarrollo de algoritmos existen tres tipos principales de estructuras repetitivas, en el caso de *PSeInt* son las siguientes:

- Mientras…Hacer
- Repetir…Hasta Que
- Para…Hasta…Con Paso

Cada una de las mencionadas estructuras repetitivas tiene características particulares y se utilizan según el tipo de repetición que se necesite.

Estructura Mientras...Hacer

La estructura de control **Mientras** permite la ejecución repetitiva de un conjunto de instrucciones siempre que se cumpla una condición lógica determinada. Esta condición se evalúa **antes** de cada iteración, lo que significa que el bloque de código se ejecutará únicamente mientras la condición sea **verdadera**.

Cuando la condición deja de cumplirse, es decir, cuando se evalúa como falsa, el ciclo se interrumpe y el flujo del programa continúa con la siguiente instrucción fuera del Mientras...Hacer.

La sintaxis de la estructura repetitiva **Mientras...Hacer** es la siguiente:

```
Mientras (condición) Hacer
   // Bloque de instrucciones
FinMientras
```

El momento de evaluar la condición es **antes** de ejecutar el bloque, lo que significa que, si la condición es falsa desde el principio, el ciclo no se ejecutará ni una sola vez.

Se presenta a continuación un ejemplo que pone en práctica la estructura repetitiva **Mientras...Hacer**.

Pasos FitLife

La app *FitLife* necesita una actualización que motive más a moverse a sus usuarios. Diseña un algoritmo que permita registrar los pasos que da cada usuario hasta lograr la meta.

Estructura repetitiva
mientras...hacer

¿Sabías qué...?

Un bucle infinito ocurre cuando la **condición de salida nunca se cumple**, y el programa sigue repitiéndose sin fin.

¿Sabías qué...?

Las estructuras repetitivas permiten que una computadora haga en segundos lo que a ti te tomaría horas.

Para saber más...

Accede al video Estructura repetitiva Mientras en *PSeInt*, para ampliar la explicación del tema. Hazlo escaneando el Código QR.

Entrada
Primero un entero que simboliza la meta en a cumplir en cantidad de pasos.
Varias líneas con un entero que corresponden a los pasos dados cada día.

Salida
Un mensaje que indicará que la meta fue alcanzada y la cantidad de días que tardó en lograrla.

Mensaje: "¡Meta alcanzada en *<número de días>* días!"

Entrada	Salida
¿Cuál es tu meta de pasos? 2000 Pasos día 1: 125 Pasos día 2: 350 Pasos día 3: 634 Pasos día 4: 500 Pasos día 5: 360 Pasos día 6: 800	¡Meta alcanzada en 6 días!

Algoritmo en *PSeInt*

```
Algoritmo FitLife
   Definir dias,meta,pasosDia,acumulado Como Entero

   dias <- 1
   meta <- 0
   pasosDia <- 0
   acumulado <- 0

   Escribir "¿Cuál es tu meta de pasos?"
   Leer meta

   Mientras (acumulado < meta) Hacer
      Escribir "Pasos día ",dias, ":"
      Leer pasosDia
      acumulado <- pasosDia + acumulado
      dias <- dias + 1
   FinMientras

   Escribir "¡Meta alcanzada en ",dias-1," días!"
FinAlgoritmo
```

Ejercitando mis conocimientos

Fortalece tu dominio de la estructura repetitiva mientras...hacer elaborando la siguiente actividad:

1. Descarga el archivo *PDF* escaneando el código QR donde encontraras un problema donde aplicaras el uso de **Estructura Repetitiva Mientras...Hacer**.

2. Genera el algoritmo en *PSeInt*, y una vez terminado, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el algoritmo creado en *PSeInt* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P2_E05**.

4. Comparte ambos archivos con tu profesor para recibir retroalimentación por el medio que acuerden.

▶ Estructura Repetir...Hasta Que

La estructura **Repetir...Hasta Que** cumple una función similar a la del ciclo **Mientras**, con la diferencia de que **evalúa la condición al final** de cada iteración. Esto implica que **el bloque de instrucciones se ejecuta al menos una vez**, sin importar el valor inicial de la condición.

En términos de lógica, podría decirse que **Mientras** es una repetición controlada por la condición al inicio, y **Repetir...Hasta Que** es una repetición controlada por la condición al final.

La sintaxis de la estructura repetitiva **Repetir...Hasta Que** es la siguiente:

```
Repetir
   // Bloque de instrucciones
Hasta Que (condición)
```

El ciclo se repetirá **mientras la condición sea falsa**, y se detendrá **cuando la condición se vuelva verdadera**.

Con el fin de ilustrar de manera más precisa el uso de estructura repetitiva **Repetir... Hasta Que**, se propone la resolución del siguiente ejercicio. En este caso, se busca validar la contraseña del usuario.

▶ *PIN de seguridad UltraApp*

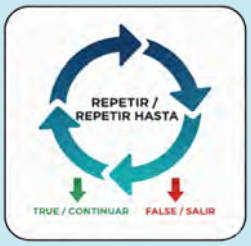
La app *UltraApp* es una aplicación que ayuda a mantener información protegida con un PIN de 4 dígitos. La app solo se desbloquea si el usuario ingresa el PIN correcto.

Como buen estudiante de pensamiento computacional quieres demostrar que puedes diseñar un algoritmo que sea eficiente para entender como funciona la aplicación *UltraApp*.

Entrada
Una cadena de caracteres que simboliza el nombre del usuario.
Un numero entero que es el NIP que permite bloquear la información
Un numero entero que es el código que se está ingresando para desbloquear.

Recurso digital

Escanea el código QR para descargar el archivo del problema de la estructura repetitiva mientras…hacer.



Estructura repetitiva
repetir...hasta

Para saber más...

Escanea el código QR y observa el video Estructura de control Repetir hasta en *PSeInt*.

Salida

Mensaje de acceso concedido: "Acceso concedido, Bienvenido <usuario> a *UltraApp!*"

Entrada	Salida
¿Cuál es tu nombre de usuario? Brithany Introduce el NIP secreto: 0212 Accede a UtraApp NIP: 3405 NIP: 0212	Acceso concedido, Bienvenid@ Brithany a *UltraApp!*

Algoritmo en *PSeInt*

```
Algoritmo UltraApp
	Definir NIP,NIP2 Como Entero
	Definir nombre Como Caracter

	NIP <- 0
	NIP2 <- 0
	Nombre <- ""

	Escribir "Cuál es tu nombre de usuario?"
	Leer usuario
	Escribir "Introduce el NIP secreto:"
	Leer NIP
	Escribir "Accede a UltraApp"

	Repetir
		Escribir "NIP: "
		Leer NIP2
	Hasta Que NIP=NIP2
	Escribir "Acceso concedido, Bienvenid@ ",nombre," a
UltraApp"
FinAlgoritmo
```

Recurso digital

Escanea el código QR para descargar el archivo del problema de la estructura repetir…hasta.

Ejercitando mis conocimientos

Para reforzar tu aprendizaje realiza de manera individual y con la guía de tu profesor la siguiente actividad:

1. Descarga el archivo *PDF* escaneando el código QR donde encontraras un problema donde aplicaras el uso de la **Estructura Condicional Repetir…Hasta**.

2. Genera el algoritmo en *PSeInt*, y una vez terminado, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el algoritmo creado en *PSeInt* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P2_E06**.

4. Comparte ambos archivos con tu profesor para recibir retroalimentación por el medio que acuerden.

Estructura Para...Hasta...Con Paso

La estructura **Para** se utiliza cuando se sabe **de antemano cuántas veces debe repetirse un conjunto de instrucciones**. Es ideal para recorrer un rango numérico o realizar cálculos que implican un contador definido.

La estructura **Para** incluye tres elementos clave:

1. **Variable de control:** variable que toma valores consecutivos.
2. **Límite inicial y final:** determinan desde qué valor comienza y hasta cuál termina.
3. **Paso:** indica el incremento o decremento entre una iteración y otra (por defecto es 1).

La sintaxis de la estructura repetitiva **Para** es la siguiente:

```
Para variable ← inicio Hasta fin Con Paso incremento Hacer
   // Bloque de instrucciones
FinPara
```

El bloque de instrucciones se ejecuta mientras la variable esté dentro del rango definido. El ejemplo que se muestra a continuación evidencia la aplicación de la estructura repetitiva **Para...Hasta**.

Likes por hora - Influencer Simulator

Te decidiste a convertirte en un creador de contenido y deseas analizar el rendimiento de tus publicaciones en redes sociales.

Durante varios días anotas el porcentaje de *likes* que obtuviste cada día respecto al número de vistas. Pero, necesitas calcular el promedio general de *likes* de todos esos días para conocer tu desempeño. Por eso decides crear un algoritmo que pida cuántos días deseas evaluar y luego solicite el porcentaje de *likes* por día.

Finalmente, mostrará el promedio general de *likes* redondeado a un decimal y un mensaje que indique si su rendimiento fue excelente, aceptable o necesita mejorar.

Entrada
Un número entero que representa los días que se van a evaluar, seguido de la cantidad de *likes* para cada uno de los días sleccionados por medio de números enteros que corresponden al porcentaje de *likes* obtenidos en cada día.

Salida
Un mensaje con el promedio de *likes*.
Mensaje: "Promedio general de *likes* <promedio>"

Entrada	Salida
¿Cuántos **días** deseas evaluar? 5 Porcentaje de *likes* el **día** 1: 85 Porcentaje de *likes* el **día** 2: 78 Porcentaje de *likes* el **día** 3: 92 Porcentaje de *likes* el **día** 4: 87 Porcentaje de *likes* el **día** 5: 80	Promedio general de *likes*: 84.40

Estructura repetitiva **Para**

Para saber más...

Accede al video Estructura de control Para en *PSeInt*, para ampliar la explicación del tema. Hazlo escaneando el Código QR.

¿Sabías qué...?

Dentro de las estructuras repetitivas puedes usar instrucciones como *break* o *continue* para controlar mejor el flujo de repetición.

Algoritmo en *PSeInt*

```
Algoritmo PromedioLikes

	Definir dias, i Como Entero
	Definir likes, promedio Como Real
	dias <- 0
	likes <- 0
	promedio <- 0
	i <- 0

	Escribir “¿Cuántos días deseas evaluar?”
	Leer dias

	Para i <- 1 Hasta dias Con Paso 1 Hacer
		Escribir “Porcentaje de likes dia ”,i,“: ”
		Leer likes
		promedio <- promedio + likes
	FinPara

	promedio <- promedio / dias

	Escribir “promedio general de likes: ”,promedio

FinAlgoritmo
```

Recurso digital

Escanea el código QR para descargar el archivo del problema de la estructura para…hasta.

Ejercitando mis conocimientos

Refuerza tu comprensión sobre el manejo de estructuras repetitivas **para…hasta** desarrollando el ejercicio de manera individual y con la guía de tu profesor:

1. Descarga el archivo *PDF* escaneando el código QR donde encontraras un problema donde aplicaras el uso de la **Estructura Repetitiva Para…Hasta**.

2. Genera el algoritmo en *PSeInt*, y una vez terminado, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el algoritmo creado en *PSeInt* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P2_E07**.

4. Comparte ambos archivos con tu profesor para recibir retroalimentación por el medio que acuerden.

La estructura de control repetitiva **Para** se utiliza en una amplia variedad de situaciones y tiene múltiples aplicaciones. En el ejemplo anterior se empleó para generar una secuencia de números con un incremento específico. Además, es útil para recorrer los elementos de un arreglo, lo que simplifica tareas de lectura, procesamiento y escritura de datos (el concepto de arreglo se trabajará en una progresión posterior).

Además, también puede utilizarse en combinación con estructuras de control condicionales, como se ha analizado anteriormente.

El dominio de las **estructuras de control** es un pilar fundamental para todo aquel que se inicia en el área de la programación.

Ventajas de combinar diferentes estructuras de control

1. Permiten crear programas más completos, capaces de crecer y adaptarse a nuevas funciones.
2. Mejoran la organización del código, lo que facilita su lectura, mantenimiento y modificación.
3. Favorece la reutilización de instrucciones, ahorrando tiempo en tareas similares.
4. Hacen posible la toma de decisiones dentro del programa, de acuerdo con la información proporcionada por el usuario.
5. Facilitan la ejecución repetida de acciones sin necesidad de escribir múltiples veces las mismas instrucciones.

Las estructuras de control son fundamentales para desarrollar programas interactivos, como sistemas de gestión, videojuegos o simulaciones.

El aprendizaje mediante *PSeInt* proporciona una base sólida, ya que prioriza la lógica algorítmica y contribuye a que se desarrolle una mentalidad analítica antes de enfrentarse a lenguajes de programación formales.

Ejercitando mis conocimientos

De manera colaborativa en clase presencial, realicen lo siguiente:

1. En un documento de *MS Word* elaboren una tabla comparativa, donde analicen y comparen las tres estructuras de control principales:

- Estructuras secuenciales
- Estructuras condicionales
- Estructuras repetitivas

Para cada estructura, incluyan los siguientes aspectos:

Aspectos para comparar	Descripción
Uso o propósito principal	¿Para qué tipo de problema se utiliza?
Funcionamiento lógico	¿Cómo se ejecutan las instrucciones en esta estructura?
Ejemplo de problema donde se aplica	Describan una situación cotidiana o algorítmica donde se usaría.
Sintaxis básica en *PSeInt*	Escribe la estructura correcta en *PSeInt*.
Ventajas y limitaciones	Comentar cuándo es más eficiente o cuándo no conviene usarla.

2. Agreguen al final de la tabla una pequeña conclusión donde reflexionen porque consideran que es necesario el uso de estructuras de control en el diseño de algoritmos para resolver problemas.

3. Guarden el documento usando en el nombre del archivo el número de equipo seguido de **_PC_P2_E08**

4. Compartan el documento con su profesor por el medio que hayan acordado.

Concretando mis conocimientos

Es tiempo de demostrar tu aprendizaje de los temas de Algoritmia en IDE, reúnete con tu equipo de trabajo y de manera colaborativa realicen lo siguiente:

1. Inicia un nuevo algoritmo en *PSeInt* que resuelva el siguiente problema: Cerebro Neuronal – Entrenando a mi IA.

La empresa *NeuraMind* te ha invitado a participar en la calibración de su nuevo modelo de inteligencia artificial llamado *NeuroCore*. Este modelo aprende a reconocer patrones visuales, y su nivel de precisión mejora con cada ronda de entrenamiento. Sin embargo, los científicos notaron tres comportamientos importantes durante las pruebas:

- **Avance constante:** en cada ronda, el modelo gana cierta cantidad de precisión que siempre está relacionada con una tasa base de aprendizaje establecida por los ingenieros.

- **Progreso acumulativo:** mientras más rondas se completan, la IA mejora más rápido porque "aprende a aprender".

- **Fatiga del sistema:** a pesar de la mejora continua, en cada entrenamiento el sistema pierde un poco de rendimiento por la sobrecarga de datos (0.5 cada ronda).

Con base en estos tres factores, tu equipo deberá descubrir y proponer una fórmula matemática que modele el aumento de la precisión del modelo de IA en cada ronda, partiendo de un valor inicial del 50% de precisión.

El modelo se considerará listo para implementarse si la precisión es mínima de 90 %.

Entrada
Deberán recibirse tres datos:

- El nombre del modelo de IA.
- Un número real que corresponde a la tasa base de aprendizaje.
- Un entero que es la cantidad de rondas de entrenamiento.

Salida
- Se deberá mostrar un mensaje que indique el nombre del modelo y que el entrenamiento ha iniciado.
- Después un mensaje por cada ronda de entrenamiento con la precisión actual de la ronda.
- Por último, un mensaje que indique si el modelo está listo para usarse o se recomienda continuar ajustando parámetros.

Entrada	Salida
• Ingresa el nombre del modelo de IA: *OrionNet* • Ingresa la tasa base de aprendizaje (%): 3 • Ingresa el número total de rondas de entrenamiento: 5	• Modelo: *OrionNet* Entrenamiento de IA iniciado... • Ronda 1 - Precisión actual: 52.5% • Ronda 2 - Precisión actual: 58% • Ronda 3 - Precisión actual: 66.5% • Ronda 4 - Precisión actual: 78% • Ronda 5 - Precisión actual 92.5% • Entrenamiento exitoso: El modelo está listo para usarse.
• Ingresa el nombre del modelo de IA: *AlphaNet* • Ingresa la tasa base de aprendizaje (%): 2.5 • Ingresa el número total de rondas de entrenamiento: 4	• Modelo: *AlphaNet* Entrenamiento de IA iniciado... • Ronda 1 - Precisión actual: 52.5% • Ronda 2 - Precisión actual: 56.5% • Ronda 3 - Precisión actual: 63.5% • Ronda 4 - Precisión actual: 73% • Entrenamiento incompleto: se recomienda continuar ajustando parámetros

2. Una vez terminado el algoritmo ejecuta y comprueba su correcto funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el algoritmo creado en *PSeInt* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P2_CMC**.

4. Comparte con tu profesor por el medio que indique tu algoritmo probado y listo para recibir evaluación.

Instrumento de evaluación

Revisa la siguiente lista de cotejo para que conozcas los criterios con los que tu profesor evaluará tu programa en *PSeInt*.

Indicador	Si	No	Puntos
El algoritmo solicita correctamente los tres datos de entrada (nombre, tasa de aprendizaje, rondas).			1
Se muestra un mensaje inicial indicando el nombre del modelo y el inicio del entrenamiento.			1
Se calcula correctamente la precisión en cada ronda considerando los tres factores: avance constante, progreso acumulativo y fatiga.			3
Se muestra la precisión actual en cada ronda con formato claro y comprensible.			2
Se incluye una condición final que evalúa si el modelo está listo (≥ 90%) o necesita ajustes.			2
El código está bien estructurado, con buena indentación y uso adecuado de variables.			1

Demostrando mi aprendizaje

Para demostrar tu aprendizaje conceptual referente a los temas abordados en esta progresión, realiza la actividad interactiva, ingresa a ella escaneando el código QR

Valorando mi aprendizaje

La evaluación es un proceso continuo de formación, útil para recabar evidencias sobre el logro de los aprendizajes, con oportunidad de retroalimentación y mejora de los resultados.

En este apartado se presentan algunas actividades e instrumentos, que te guían en la valoración de los aprendizajes que adquiriste progresivamente en las primeras dos secuencias didácticas. Responde honestamente a cada una de ellas.

Reflexionando lo que aprendí

Contesta las siguientes preguntas y reflexiona sobre tu desempeño en estas dos progresiones.

- ¿Cuál de las cuatro fases del pensamiento computacional (descomposición, reconocimiento de patrones, abstracción y diseño de algoritmos) te resultó más fácil de comprender y aplicar? ¿Por qué?

- Imagina que debes enseñarle a un compañero qué es un algoritmo. ¿Qué ejemplo sencillo usarías y por qué lo elegirías para facilitar su comprensión?

- ¿Cómo te ayudó *PSeInt* a visualizar o comprender la lógica detrás de las estructuras secuenciales, condicionales o repetitivas? Explica con un ejemplo.

- Después de trabajar estos temas ¿qué crees que te falta reforzar o seguir practicando en relación con la creación y análisis de algoritmos? Explica cómo planeas mejorar.

Actividad alternativa

Resuelve la siguiente actividad alternativa para reforzar tus aprendizajes e incrementar tu evaluación sumativa.

1. Demuestra tu aprendizaje en el tema resolución de problemas estructurados creando un video acerca de las fases del pensamiento computacional.

2. Explica cada una de las fases de manera clara.

3. Puedes guiar el video resolviendo algún problema sencillo y cotidiano.

4. Publícalo y envía en enlace a tu profesor para que observe el video y evalúe tu actividad.

Autoevaluación

La autoevaluación es un mecanismo de autocontrol que te ayuda a regular tu aprendizaje. Marca con una √ la columna que corresponda a tu nivel de dominio en los aspectos de aprendizaje en cada meta.

Metas	Criterios	Nivel de dominio		
		Sí lo logro	En proceso	Aún no lo logro
Identifica los principios del pensamiento computacional, su descomposición, abstracción y patrones para diseñar, implementar y evaluar algoritmos de problemas de su vida cotidiana.	Identifico los principios del pensamiento computacional.			
	Aplico cada una de las fases para resolver problemas cotidianos.			
Representa la solución de problemas mediante pensamiento algorítmico seleccionando métodos, diagramas o técnicas.	Organizo los pasos del algoritmo de manera clara y lógica.			
	Represento soluciones mediante pseudocódigo y diagramas de flujo.			
Aplica lenguaje algorítmico utilizando medios digitales para resolver situaciones o problemas del contexto.	Traduzco mis soluciones algorítmicas a pseudocódigo en un entorno digital.			
	Ejecuto mis algoritmos en *PSeInt* para validar su funcionamiento.			
Identifica situaciones de la vida cotidiana que pueden resolverse de manera más eficiente utilizando secuencias y ciclos.	Reconozco problemas que pueden resolverse con secuencias y ciclos.			
	Selecciono la estructura de control adecuada (secuencia, condicional o repetitiva) en la resolución de problemas.			
Comprueba la lógica y funcionamiento de algoritmos para representar sus soluciones mediante IDE corrigiendo errores y optimizando el código.	Compruebo el funcionamiento de mis algoritmos mediante la ejecución en el IDE.			
	Identifico errores de sintaxis, de lógica y de ejecución en *PSeInt*.			
Lo mejor que aprendí fue:				
Lo que necesito reforzar es:				

Calificación que doy a mi desempeño:	Excelente	Satisfactorio	En desarrollo	Inicial

Coevaluación

Evalúa el desempeño general de tu equipo de trabajo durante el desarrollo de las actividades de aprendizaje colaborativas. Coloca el valor correspondiente en la columna Evaluación y suma para conocer el resultado del trabajo por equipo.

Buen trabajo (3)	Algo nos faltó (2)	Debemos mejorar (1)	Evaluación
Organizamos el trabajo estipulando tareas, prioridades y plazos.	Se organizó el trabajo, pero no se estipularon tareas, prioridades o el plazo de entrega final.	No hubo organización para realizar nuestros trabajos.	
Cumplimos cada uno con las tareas asignadas en el plazo estipulado.	Casi todos los miembros del equipo cumplimos con las tareas asignadas y el plazo estipulado; teniendo que resolver lo que a otros les fue encomendado.	Un solo miembro del equipo realizó todos los productos.	
Todos participamos activamente en la elaboración de los productos.	Casi todos los miembros del equipo participamos activamente en la elaboración de los productos.	No hubo participación de los miembros del equipo en la elaboración de los productos.	
La calidad de los productos que elaboramos fue la adecuada para su entrega.	La calidad de los productos que elaboramos fue en su mayoría la adecuada para su entrega.	No se cumplió con la calidad adecuada de los productos para su entrega.	
		Total	___ de 12

Progresión 3

Programación estructurada en C++: Estructuras de control

Codifica instrucciones en un lenguaje de programación estructurada, empleando estructuras de control secuenciales y repetitivas para determinar el orden lógico y eficiente en que se ejecutan en la resolución de problemas.

Tiempo estimado: 9 horas

Tus metas serán:

- Distinguir la sintaxis básica de *C++* y la utilidad de las estructuras de control para organizar la ejecución de instrucciones.
- Representar soluciones a problemas cotidianos y académicos mediante algoritmos que incorporan estructuras de control secuenciales y repetitivas.
- Codificar, compilar y ejecutar programas en *C++* validando su funcionamiento y corrigiendo errores en el uso de estructuras de control.

Recuperando lo que sabemos

Este cuestionario es de recuperación de conocimientos previos, es útil para identificar tus saberes y habilidades y cómo los relacionas con la realidad, además te ayudará a comprender mejor los temas de esta secuencia. No es necesario que conozcas los términos técnicos; lo importante es expresar cómo entiendes o aplicarías cada situación, haz tu mejor esfuerzo y detecta aquellos aspectos que no conoces o dominas para enfocar tu estudio.

1. ¿Qué entiendes por programar y por qué crees que es importante en la actualidad?

_____

_____

_____

2. ¿Qué problemas cotidianos crees que podrían resolverse mediante la programación?

_____

_____

3. ¿Cómo imaginas que se comunica una computadora con el programador para ejecutar instrucciones?

_____

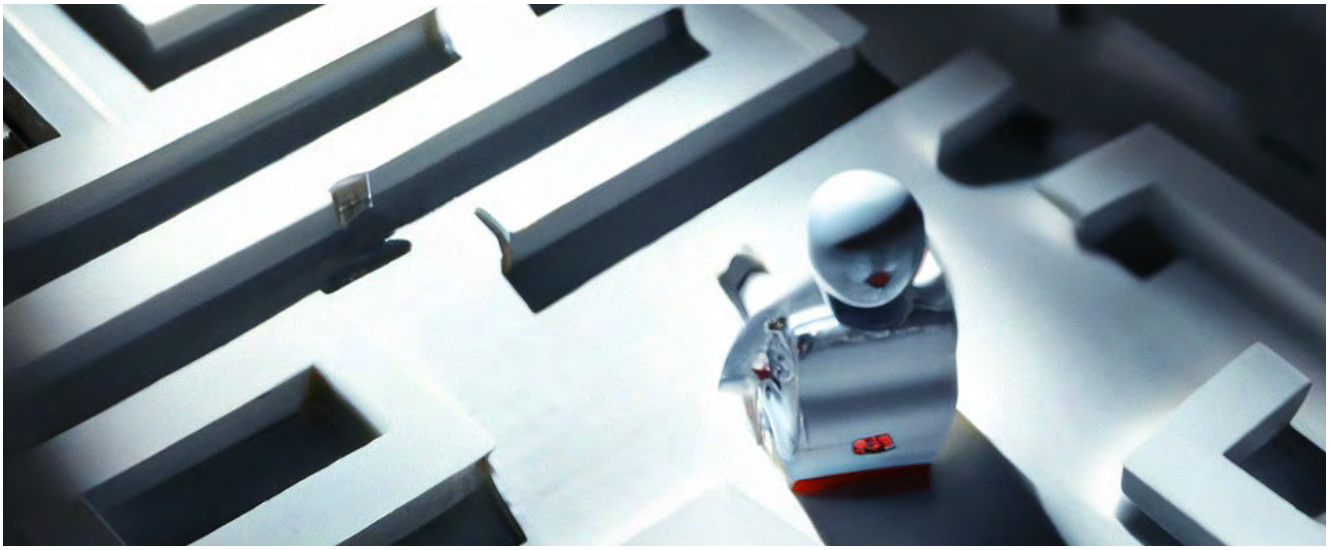
_____

4. ¿Qué opinas de la relación entre lógica y programación? ¿Por qué crees que son importantes juntas?

_____

_____

_____



Reactivando mis conocimientos

Imagina que eres un pequeño robot que solo entiende un conjunto limitado de instrucciones. Tu misión es salir del laberinto de la imagen, utilizando la menor cantidad de instrucciones posibles.

¡Pistas importantes!

1. Asume que el robot siempre empieza mirando hacia arriba desde la posición de INICIO.

2. El laberinto tiene puntos en su interior.

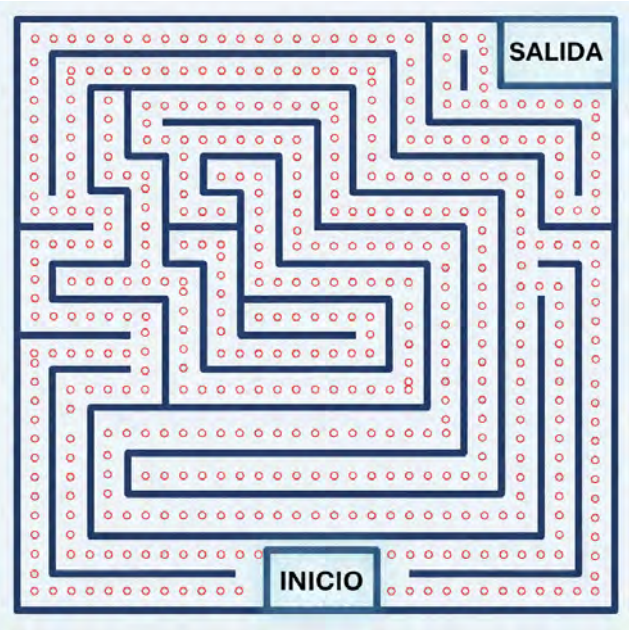
3. Cada vez que uses `AVANZAR_1_PASO`, el robot se moverá de un punto a otro.

4. Si detectas que tienes que hacer la misma acción varias veces seguidas, puedes usar la instrucción `REPETIR X VECES (acción)` para ahorrar órdenes.

Escribe en tu cuaderno de notas la secuencia de instrucciones para que el robot salga del laberinto, comenzando en INICIO y terminando en SALIDA. Intenta ser lo más eficiente posible usando `REPETIR`.

Instrucciones disponibles para el robot:

- `AVANZAR_1_PASO` - Cada paso te lleva al siguiente punto.
- `GIRAR_DERECHA` - Gira 90° a la derecha, manteniendo tu posición actual.
- `GIRAR_IZQUIERDA` - Gira 90 a la izquierda, manteniendo tu posición actual.
- `REPETIR_VECES (acción)` Por ejemplo `REPETIR 3 VECES (AVANZAR_1_PASO)`



3.1 Lenguajes de programación

¿Sabías qué...?

En este símbolo universal, la línea vertical representa encendido, mientras que el círculo, el apagado; convención que en 1973 se estandarizo para los dispositivos electrónicos basándose en los símbolos binarios 1 – Encendido y 0 – Apagado.

Relaciónalo con...

En los inicios de la computación, las tarjetas perforadas, hechas de cartulina, eran el principal medio para almacenar y suministrar datos e instrucciones a una computadora. Su uso era en una forma de código binario físico; en una posición especifica, un agujero representaba un '1' o verdadero y la ausencia de agujero un '0' o falso. Los programadores creaban sus programas perforando tarjetas, una por una, luego las apilaban en orden correcto.

En tanto que el pensamiento computacional es la habilidad cognitiva para analizar y resolver problemas de manera lógica y estructurada, la programación es la implementación técnica de una solución mediante un lenguaje de programación. En otras palabras, el pensamiento computacional es el proceso mental para idear la solución y la programación es el acto de materializar esa solución en código.

Programación es la acción de traducir un algoritmo a instrucciones que una computadora pueda ejecutar. Su objetivo es crear un *software* funcional que resuelva el problema planteado. Para ello se requiere de conocimientos específicos de sintaxis, estructuras de datos y entornos de desarrollo. Estas instrucciones, reglas y sintaxis que permitan a los humanos comunicarse con las computadoras para indicarles qué tareas deben realizar, se conocen como **lenguajes de programación**.

Entre los programas más populares que los jóvenes usan hoy día están las distintas redes sociales, el *software* que usan para comunicarse, los que usan como medio de entretenimiento para ver películas en *Streaming* o los videojuegos, aunque también se puede hablar de plataformas educativas y productivas donde organizar tareas o aprender algún lenguaje mediante ejercicios interactivos.

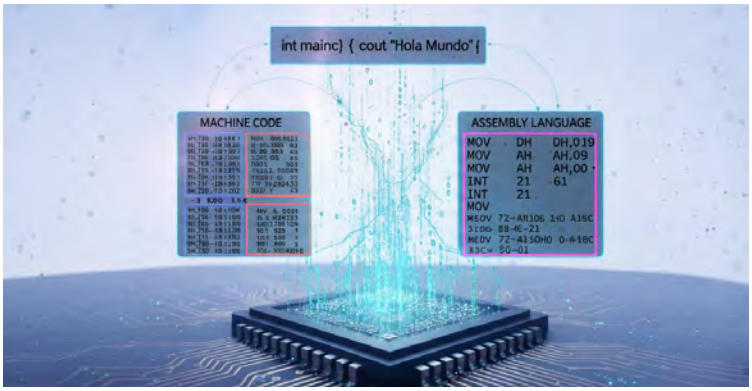
Historia de los lenguajes de programación

La programación ha evolucionado junto con la tecnología. En los inicios de la computación, por allá en la década de 1940 y 1950, se comenzó con las tarjetas perforadas y los programas se escribían directamente en **lenguaje máquina**, que usa y entiende solo el sistema binario basado en 2 dígitos: sus símbolos son: 0s (ceros) y 1s (unos), físicamente son apagado y encendido.

La combinación de estos dígitos es lo que usa este lenguaje para representar los valores, por ejemplo el número 1001101, en sistema decimal es el número 77. El lenguaje máquina no entiende de números decimales y letras directamente, sino que las interpreta como secuencias de números binarios mediante el sistema ASCII. Programar en este lenguaje es un método poco práctico y propenso a errores.

Lenguaje máquina con el texto "Hola mundo".

Con el tiempo, se buscó hacer la programación más fácil para los humanos, lo que llevó al surgimiento de los lenguajes de bajo nivel, muy cercanos al hardware de la computadora, rápidos pero difícil de escribir y mantener; entre ellos está el popular lenguaje ensamblador, que permitía usar palabras simbólicas, como *MOV, ADD, SUB* en lugar de números binarios. Sin embargo, seguían siendo de bajo nivel, muy cercanos al lenguaje de la computadora y poco intuitivos para las personas.



Transición de lenguaje máquina a instrucciones de lenguaje ensamblador.

La característica principal de estos lenguajes son su alto grado de abstracción, lo que significa que oculta los detalles de la arquitectura del hardware, permitiendo a los programadores centrarse en la lógica del problema a resolver en lugar de detalles de bajo nivel como la gestión de la memoria.

Algunos de los más influyentes fueron: *Fortran, Cobol, Basic, C* y *C++*. Actualmente lenguajes como *Pythom, Java* y *JavaScript* son mayormente usados por tener una sintaxis clara permitiendo crear aplicaciones robustas o para el desarrollo en web, mientras que Basic es fácil de usar y aprender.

Posterior a los lenguajes de alto nivel, se tomó lo mejor de *C* y se le añadió una forma de programar llamada **Programación Orientada a Objetos** (OOP).

Al principio se le llamó *C* con Clases, pero luego se rebautizó como *C++*, ofreciendo un balance entre el control de bajo nivel y las abstracciones de alto nivel, ganándose el termino de lenguaje medio por su facilidad de lenguaje y el poder que da de manipular la memoria.

Conceptos clave

ASCII. (American Estándar Code for Information Interchange). Sistema de codificación estándar para caracteres que permite a las computadoras representar texto. Utiliza una combinación de 7 u 8 valores (bits) para representar 128 o 256 caracteres, respectivamente. Por ejemplo, la palabra ´Hola´ en lenguaje máquina se escribe como 01001000 01101111 01101100 01100001 y cuyo código ASCII es 72 111 108 108 97.

Para saber más...

Accede a la presentación interactiva Lenguajes de programación y conoce más características. Hazlo escaneando el Código QR.

Clasificación de los lenguajes

Existen varias formas de clasificar los lenguajes:

Por nivel de abstracción:

- **Bajo nivel.** Como ensamblador y lenguaje máquina. Si bien son rápidos también son difíciles de escribir y no son portables, es decir, un código de ensamblador para el celular no funciona en una computadora personal.
- **Alto nivel.** Entre ellos *Python, Java, JavaScript, C++* y *C#*. Son más fáciles de leer y escribir, además son portables.

Por paradigma de programación:

Esta clasificación se refiere a la filosofía de cómo programar.

- **Imperativo.** En él se le dice a la computadora qué hacer y cómo hacerlo, paso a paso, por ejemplo la programación estructurada.
- **Orientado a objetos.** Usado para crear objetos que tienen datos, es decir atributos; también tienen funciones, métodos.
- **Funcional.** Se basa en el uso de funciones matemáticas puras.

Por ejecución:

- **Compilados.** Se escribe el código fuente y luego un programa especial llamado compilador lo traduce todo de una vez a lenguaje máquina, creando un archivo ejecutable, haciendo que la ejecución sea súper rápida.
- **Interpretados.** En él, un programa llamado intérprete lee el código línea por línea y lo ejecuta al momento. Es más lento, pero a menudo más flexibles para hacer pruebas rápidas.

Clasificación de los lenguajes de programación.

Estudiando

Dedica un tiempo a la lectura de las páginas y observar recursos didácticos correspondientes a los temas de **Programación estructurada en *C++*: Estructuras de control**. Realizar esta tarea, te facilitará el aprendizaje y realizar las actividades que el profesor guiará en las siguientes sesiones. Apóyate en alguna estrategia de lectura que te ayude a mejorar la comprensión lectora.

Editores de código

El código de programación no puede ser escrito en un procesador de texto común como la aplicación *Microsoft Word*, se necesita un editor de texto plano, como el **editor de código**, herramienta donde los desarrolladores escriben y organizan los programas, pues está diseñado para facilitar el desarrollo de software mediante funciones específicas para programar. Sus funciones más comunes son:

- Resaltado de sintaxis coloreando palabras clave, variables y operadores.
- Autocompletado de código, mientras se escribe sugiere comandos y ayuda para evitar errores.
- Compilación integrada, permitiendo transformar el código fuente en un ejecutable.
- Depuración, esto es, que ejecuta el programa paso a paso para encontrar fallos.
- Gestión de proyectos, esto significa que el editor organiza varios archivos lo que facilita la estructura del proyecto.
- Consola integrada que muestra resultados inmediatos.

Tipos de editores de código

1. Editores básicos con soporte para programación. Sus características son ser ligeros y útiles para pequeños programas; pero tienen la desventaja de no incluir un compilador. Entre ellos, están el *Notepad++*, *Sublime Text* y *Vim*.

2. Entornos de Desarrollo Integrados (IDE). Estos editores son más completos porque incluyen editor, compilador, depurador y herramientas profesionales. Son ideales para programación estructurada porque simplifican el proceso de codificación, incluso en la jerga informática se les llama navaja suiza. Ejemplos de este tipo de editor son *Code::Blocks, Dev C++, Visual Studio, Eclipse CDT*.

3. Editores de código en línea (Online IDE). Estos permiten programar sin instalar nada, directamente desde el navegador, como el *Replit, JDoodle* y *OnlineGDB*.

▶ Editor de código *Code::Blocks*

Code::Blocks es un **Entorno de Desarrollo Integrado (IDE)** muy utilizado para programar en los lenguajes *C* y *C++*. Su diseño sencillo y funcional lo hace ideal para estudiantes que se inician en el mundo de la programación, además de ser gratuito y de código abierto. Es totalmente configurable y es altamente extensible, es decir, que está basado en un marco de plugins, un componente de software que extiende o añade funcionalidades a la aplicación principal sin alterar su código base.

Se caracteriza por tener:

1. Instalación sencilla. La descarga, preferiblemente la versión que incluye el compilador *MinGW*, puede ser del sitio oficial **https://www.codeblocks.org/downloads/** o bien desde el QR que está al lado. Posterior a la descarga, la instalación se ejecuta, se acepta la licencia y se sigue instrucciones manteniendo la configuración predeterminada si así se desea, si no, se configura manualmente desde Settings, donde también puede cambiarse el entorno a español, esto se hace en la opción Entorno > Ver > Internacionalización.

Conceptos clave

Editor de texto. Herramienta donde se escribe el código con colores para detectar errores fácilmente.

Compilador. Es un programa que traduce el código fuente de un lenguaje de programación de alto nivel a uno de bajo nivel para que la computadora lo entienda directamente.

Depurador. Programa que permite a los desarrolladores identificar y corregir errores en otro programa al ejecutarlo de forma controlada, además de examinar su comportamiento.

Recurso digital

Escanea el QR para descargar el instalador de la aplicación *Code::Blocks*.

Para saber más...

Observa al video Conociendo el IDE *Code::Blocks* en la estructura básica de un programa en *C++*. Accede a él escaneando el Código QR.

2. Interfaz amigable. Presenta una organización clara del entorno, con una ventana para el proyecto, un editor de archivos, una consola de salida y una barra de herramientas para compilar y ejecutar el programa. Además de ser intuitivo y facilita adaptarse rápidamente a la sintaxis de *C++*.

3. Compilador integrado. Permite compilar con un solo clic, generar archivos ejecutables y visualizar errores de compilación de forma clara. La versión más recomendada es *Code::Blocks* con el compilador *MinGW* por ser compatible con *Windows*. El IDE proporciona las herramientas de edición, compilación y depuración, y el compilador traduce el código fuente a código ejecutable.

4. Depurador visual. Una herramienta fundamental para aprender a pensar como un desarrollador, pues ejecuta el programa línea por línea, muestra valores de variables en tiempo real permitiendo identificar errores lógicos, además coloca *breakpoints* (punto de interrupción), esto es que establece un marcador en una línea específica del código donde se desea que la ejecución del programa se detenga temporalmente.

5. Resaltado de sintaxis. Colorea palabras reservadas, identificadores, operadores y comentarios facilitando la lectura del código y la detención rápida de errores.

6. Plantillas de proyectos. Al iniciar un nuevo proyecto (programa) puede elegirse hacer con *Console Application* o *Empty Project*. Con la primera opción se puede elegir C++ para que genere automáticamente el archivo main.cpp.

7. Multiplataforma. Este IDE funciona correctamente en diferentes sistemas operativos, por ejemplo puede ejecutarse en *Windows, Linux* o en *macOS*.

Pantalla de inicio del IDE *Code::Blocks*.

Flujo de Code::Blocks para la resolución sistemática del problema:

Crear un proyecto nuevo → Escribir el código → Guardar → Compilar → Ejecutar → Revisar resultados → Corregir errores → Compilar y ejecutar de nuevo

Pasos para crear un nuevo proyecto en Code::Blocks

1. Abrir *Code::Blocks*.

2. Crear un nuevo proyecto desde *Archivo* > *Nuevo* > *Proyecto* > *Consola de aplicación*.

Ventana Nuevo proyecto en *Code::Blocks*.

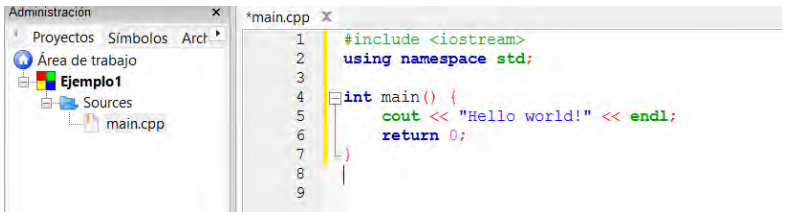
3. Seleccionar el lenguaje de programación en el que se desea codificar, puede ser *C* o *C++*.

4. Asignar el nombre y ubicación donde se guardará el proyecto. Con esto se generará una carpeta con el nombre asignado y dentro el archivo del proyecto con extensión '.cbp'.

5. Configurar el compilador: *GNU GCC Compiler* o *MinGW Compiler* (en *Windows*).

6. Abrir desde el panel lateral de Administración el archivo principal con doble clic sobre él. La aplicación le asigna el nombre 'main.cpp'.

7. Escribir o modificar el código en la ventana Editor de texto.



Panel Administración (derecha) y Editor de texto (izquierda) en *Code::Blocks*.

8. Guardar el archivo.

9. Compilar y ejecutar el programa con la tecla F9 o desde la barra superior de opciones con el comando *Build and run* (Construir y ejecutar, en entorno en idioma español).

10. Verificar errores y corregirlos. Si hay errores, la aplicación mostrará una lista en la parte inferior con mensajes como *expected ';' before '}' token*, entonces se da clic en el mensaje para ir a la línea exacta del error, se corrige y se vuelve a compilar.

Relaciónalo con...

Code::Blocks en un nuevo proyecto genera:

Archivos fuente principal (main.cpp) es donde se escribe el programa.
Archivo de proyecto (.cbp) que contiene configuración interna del proyecto como rutas y versión de compilación, archivos incluidos y parámetros de depuración.
Carpeta bin, que tiene los binarios generados, dependiendo del sistema puede haber *bin/debug* y *bin/reléase*. En ellos estará el archivo ejecutable.
Carpeta obj, (.o), son archivos objeto creados por el compilador durante la traducción.
Archivos temporales para uso interno del compilador.
Archivo ejecutable final (.exe) que se genera mediante el proceso de compilación y enlazado (*linking*) realizado por el compilador.

Ventana para configurar el Compilador en *Code::Blocks*.

3.2 Estructura básica de un programa en C++

Logo de lenguaje *C++*.

Programación estructurada

En el tipo de lenguaje de programación estructurada, los programas se diseñan de arriba hacia abajo *(top-down)* jerárquicamente, usando solo un conjunto restringido de estructuras de control en cada nivel, instrucciones secuenciales, estructuras selectivas y estructuras repetitivas. Cuando esto se hace de forma adecuada el programa resulta muy fácil de entender, depurar y modificar.

La **programación estructurada** es una forma de construir *software* de manera ordenada y clara, es como construir con bloques de *LEGO*, en lugar de tener un montón de piezas desordenadas. Los programas deben estar dotados de una estructura y escribirse de acuerdo con las siguiente reglas:

- Tener diseño modular.
- Módulos diseñados en modo ascendente.
- Codificar cada módulo utilizando los tres tipos de estructuras:
    1. **Secuencia:** ejecutar instrucciones una después de otra, como seguir una receta.
    2. **Selección:** tomar decisiones, por ejemplo, si llueve, usa paraguas; si no, usa lentes.
    3. **Iteración:** repetir tareas, como batir la mezcla de un pastel 100 veces.

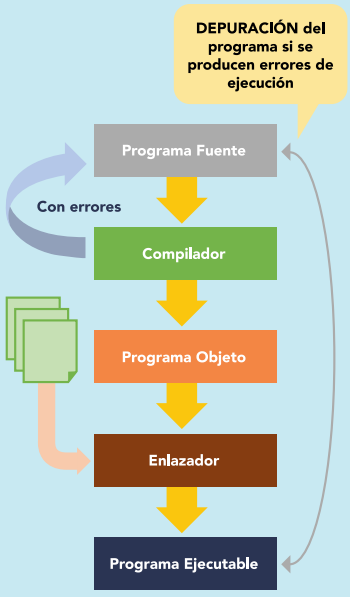


Diagrama del proceso de creación de un programa o aplicación.

Metodología para codificación de un programa

El desarrollo del software requiere atender la gestión, diseño, desarrollo e implantación para lograr su calidad. En la etapa de diseño se lleva a cabo la programación de computadoras contemplando las actividades de planeación, codificación, prueba y documentación. Para ello se precisa de una metodología que atienda el proceso de transferencia de las secuencias lógicas de un algoritmo a un determinado lenguaje de programación, lo que a su vez solicita el cumplimiento de los pasos: codificación del programa, compilación y ejecución, y verificación y depuración.

1. Codificación de un programa

Es la escritura del algoritmo en un lenguaje de programación desarrollado en las etapas precedentes. El código puede ser escrito con igual facilidad en un lenguaje o en otro. Para realizar esta conversión se deben sustituir las palabras reservadas en español por sus homónimos en inglés y las operaciones e instrucciones indicadas en lenguaje natural, expresarlas en la sintaxis del lenguaje de programación en uso.

2. Compilación y ejecución

Una vez que el algoritmo mediante un programa editor se codifica en un lenguaje, se genera un programa fuente y al colocarlo en la memoria de la computadora, es decir, se compila, se convierte el programa fuente en un archivo de programa. Si tras la compilación se presentan errores (errores de compilación) en el programa fuente, es necesario regresar a editar el programa, corregir los errores y compilar de nuevo. El proceso debe repetirse hasta que no haya errores.

La compilación sin errores da como resultado el programa objeto, este archivo aún no es ejecutable; para ello se pide al Sistema Operativo que lo enlace con las bibliotecas del compilador. Este último proceso de montaje produce el programa ejecutable, que "corre" con solo teclear su nombre desde el sistema operativo.

3. Verificación e implementación

Es el proceso de ejecución del programa con distintos datos de entrada, que determinarán si el programa tiene errores *(bugs)*. Para ello se debe hacer pruebas de datos con valores normales y extremos de entrada que comprueben los límites del programa y valores de entrada que comprueben aspectos especiales del programa.

Lenguaje C++

C++ es un lenguaje de alto nivel increíblemente poderoso y rápido que se usa para todo: sistemas operativos, como *Windows* o *macOS*, para videojuegos AAA, aplicaciones de escritorio, e incluso en la robótica y la exploración espacial.

Este lenguaje combina la programación estructurada y la orientada a objetos, convirtiéndolo en uno de los lenguajes más versátiles y utilizados en la industria. Por su capacidad para controlar eficientemente recursos de la computadora, sigue siendo uno de los lenguajes preferidos en videojuegos, *software* científico, control de *hardware*, simulaciones y sistemas operativos.

Sintaxis y elementos básicos

Así como un texto formal tiene introducción, desarrollo y conclusión, un programa está formado por partes definidas que permiten que el compilador interprete y ejecute las instrucciones del programador. En esta secuencia se describe el lenguaje *C++*, uno de los más utilizados en la enseñanza de la programación estructurada y base de muchos otros lenguajes modernos. Un programa básico en *C++* luce así:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hola, mundo!" << endl;
    return 0;
}
```

▶ **Sintaxis.** Es el conjunto de reglas que indican cómo deben escribirse las instrucciones para que el lenguajes las entienda. Por ejemplo, en *C++*, es estricta la regla de que si un carácter, símbolo o palabra clave está mal escrito o colocado en un sitio, el programa no compilará.

▶ **Directivas de preprocesador (#include).** Indica al compilador que se incluirá una biblioteca; estas funcionan como caja de herramientas que agregan funciones ya programadas listas para usarse. El ejemplo se usa el iostream, pero hay otras más completas:

Relaciónalo con...

Los videojuegos AAA son títulos de alto presupuesto, desarrollados por grandes editoriales en lenguaje *C++*; se caracterizan por sus altos costes de producción y marketing y el gran número de personas que trabajan en ellos. Por ejemplo *Grand Theft Auto V*, *Elder Ring*, *Fortnite* y *Call of Cuty: Warzone*.

Relaciónalo con...

El manipulador *endl* inserta un salto de línea en el código. Y este: >> es el operador de extracción, es decir, extrae lo que el usuario teclea y lo guarda en una variable.

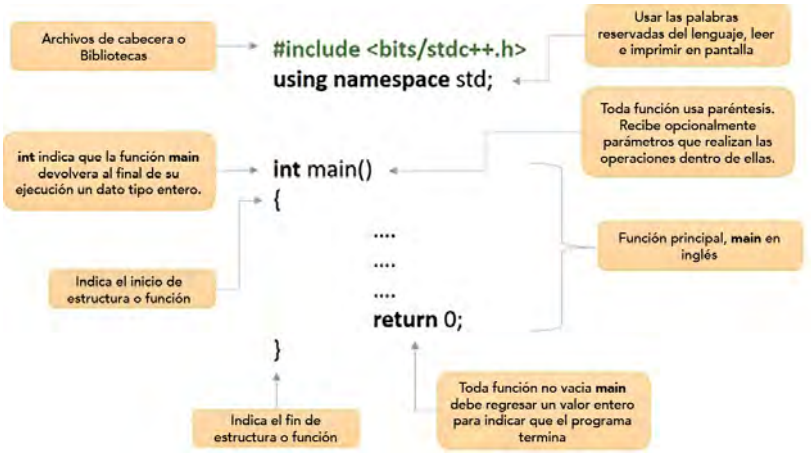
▶ **Espacio de nombres (namespace).** *C++* es un lenguaje muy grande, lleno de funciones y objetos; para evitar que sus nombres choquen entre sí, se organizan en *namespaces*. También permite usarlos sin reescribirlos en cada llamado. En el ejemplo se utiliza: *using namespace* ***std***; que contiene las herramientas esenciales:

- *cout* → salida de datos
- *cin* → entrada de datos
- *string* → manejo de textos

▶ **Función principal.** Todo programa en *C++* debe tener una función llamada ***main()*** pues es el punto de inicio donde el programa comienza a ejecutarse. Siguiendo con el ejemplo:

- *int* → indica que la función regresa un valor entero al sistema operativo.
- *return 0*; → significa que el programa terminó correctamente.
- { } → las llaves delimitan dónde empiezan y terminan las instrucciones del programa.

▶ **Instrucciones y declaraciones de sentencias.** Dentro del *main()* van todas las instrucciones a ejecutar. Cada línea termina con punto y coma (;) para indicar el final de una instrucción, si se olvida, el programa marcará error. En el ejemplo aparece: *cout << "Bienvenido a C++" << endl;*



Estructura de programa en *C++*.

¿Sabías qué...?

Cuando se usarán datos de tipo carácter (letra, número o símbolo) se usan comillas simples: ' '. Para guardar datos de cadena de texto se usan comillas dobles: " ", también debe incluirse la librería *#include <string>*.

Variables y tipo de datos

El espacio en memoria donde se almacena un dato en el programa son las **variables**. Pero para usarlas, primero se deben **declarar**, esto significa indicar su nombre y qué tipo de datos guardará.

▶ Sintaxis general de la declaración de variable es:

- Tipo nombre;
- Tipo nombre = valor;

- Reglas para nombrarlas:
  - No puede iniciar con número.
  - No deben incluir espacios.
  - Deben evitar acentos y caracteres espaciales.
  - No deben llamarse igual que las palabras clave del lenguaje.
  - Correcto → edadAlumnos , promedio_final , total2
  - Incorrecto → 2edad , promedio final , c++ , if

Tipos de datos más comunes en C++

Tipo	Descripción	Ejemplo de declaración
int	Número enteros	int edad = 17;
float	Números con decimales (precisión simple)	float pi = 3.1416;
double	Decimales de mayor precisión	double gravedad = 9.81;
char	Un solo carácter	char letra = 'A';
bool	Verdadero o falso	bool activo = true;
string	Cadenas de texto	string nombre = "Claudia";

Entrada y salida de datos

Dos procesos fundamentales para la comunicación entre un sistema de información y el mundo exterior, permitiendo que el usuario interactúe con el programa y vea el resultado de las operaciones son la entrada *(input)* que es la información que el programa recibe y la salida *(output)* es decir, el resultado que el programa envía después del procesamiento. En lenguaje *C++* se usa la sintaxis:

- *cout* →Objeto de salida de consola. Por ejemplo:
  cout << "Edad: " << edad << endl;
- *cin* →Objeto de entrada de consola, por ejemplo:
  cin >> edad;

Para saber más...

Escanea el código QR para consultar una infografía donde se explica la jerarquía de operaciones en *C++*.

Operadores en C++

Los operadores permiten realizar cálculos, comparaciones y decisiones dentro de un programa. Son fundamentales para las estructuras control. Hay cuatro tipos de operadores:

1. Operadores aritméticos.
Permiten hacer los cálculos matemáticos básicos: suma (+), resta (-), multiplicación (*), división (/), modulo o residuo (%).

2. Operadores relacionales.
Se usan para comprar valores, su resultado siempre es *true* o *false*: igual que (==), diferente de (!=), menor que (<), mayor que (>), menor o igual (<=), (>=) mayor o igual.

3. Operadores lógicos.
Permite unir comparaciones: dos datos verdaderos (AND - &&), al menos uno verdadero (OR – ll) invierte el valor de verdad (NOT - !).

4. Operadores de asignación.
Hay dos formas: asignar =; operar y asignar +=, -=, *=, /=.

```
#include <iostream>
using namespace std;

int main() {
    int edad;
    cout << “Ingresa tu edad: ”;
    cin >> edad;

    if (edad >= 18) {
        cout << “Eres mayor de edad.” << endl;
    } else {
        cout << “Eres menor de edad.” << endl;
    }
    return 0;
}
```

3.3 Estructuras de control

Las estructuras de control como su nombre indica **controlan el flujo de ejecución de un programa** y aquí es donde la programación se vuelve poderosa, con las estructuras de control selectivas y repetitivas que permiten que un programa no sea una simple secuencia lineal de instrucciones, sino que tome decisiones y repita acciones según las condiciones establecidas. Aunque, las estructuras secuenciales son fundamentales por ser la base de cualquier algoritmo.

Estructuras secuenciales

Este tipo de estructuras permiten la ejecución de instrucciones en un orden específico y lógico, su principal característica radica en que los hacen los programas más fáciles de entender, depurar y mantener por **ejecutar una instrucción tras otra**, en el orden en que aparecen en el código sin saltos ni bifurcaciones. Aunque en programas complejos se combinan con estructuras condicionales y bucles.

La secuencia es crucial para tareas simples y para definir el flujo básico de acción en cualquier programa.

Algoritmo	Código en *C++*
Inicio *Instrucción(es)* *Fin*	*int main() {* *instrucción(es);* *return 0;* *}*

El siguiente código en lenguaje *C++* es un ejemplo de estructuras secuenciales, su problema y algoritmo en *PSeInt* se encuentran en el recurso digital de al lado:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    float mouse=0,audifonos=0,tapete=0,iva=0,total=0;

    cout << "Precio del mouse: ";
    cin >> mouse;
    cout << "Precio de los audifonos: ";
    cin >> audifonos;
    cout << "Precio del tapete: ";
    cin >> tapete;

    total = (mouse + audifonos + tapete)*1.16;
    cout << "Total a pagar: " << fixed <<
setprecision(2) << total << endl;

    return 0;
}
```

Recurso digital

Escanea el QR para descargar el archivo con Ejemplos de Estructuras de control. En él se encuentran los problemas y algoritmos en *PSeInt* de todos los ejemplos de la progresión.

Para saber más...

Observa el video Estructura condicional *If* en C++ para profundizar en la explicación de tu profesor. Accede a él escaneando el Código QR.

Para saber más...

Observa el video explicativo Estructura condicional *If else* en C++ y refuerza el tema visto en clase. Accede a él escaneando el Código QR.

Estructuras condicionales

Como se vio en la secuencia de *PSeInt*, las estructuras de control condicionales o de selección se usan para que **el programa elija un camino**. El flujo puede ser de tres formas: *if, if-else, if-else-if.*

1. Condicionales simples - Si (if). Son estructuras selectivas simples que ejecutan un bloque de código solo si la condición es verdadera.

Algoritmo	Código en C++
Si (condición(es)) entonces *Instrucción(es)* *Fin del si*	*if(condición(es)){* *instrucción(es);* *}*

Ejemplo de selectiva simple. Tanto como el problema como el algoritmo en *PSeInt* están en el recurso digital de la página 71.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int edad = 0;

    cout << "¿Cuál es tu edad? ";
    cin >> edad;
    if ( edad > 15 ){
        cout << "Acceso permitido" << endl;
    }
    return 0;
}
```

2. Condicionales dobles - Si... si no... (if-else). Llamadas también selectivas dobles. Ejecuta un bloque si la condición es verdadera y un bloque diferente si es falsa.

Algoritmo	Código en C++
Si (condición(es)) entonces *Instrucción(es)* *De lo contrario Si (condición(es)) entonces* *Instrucción(es)* *Fin del si*	*if(condición(es)){* *instrucción(es);* *}else{* *instrucción(es);* *}*

Ejemplo de selectiva doble. El problema y el algoritmo en *PSeInt* se encuentran en el recurso digital de la página 71.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    float precio = 0, total = 0;
    char credencial = ' ';
    cout << "Cuál es el precio del libro? ";
    cin >> precio;
    cout << "¿Tienes credencial de estudiante? (S=si,
N=no): ";
    cin >> credencial;
    if (credencial == 's' || credencial == 'S') {
        total = precio * 0.9;  //Aplicar 10% de desc.
    } else {
        total = precio;
    }
    cout << "Total a pagar: " << fixed <<
setprecision(2) << total << endl;
    return 0;
}
```

Ejercitando mis conocimientos

Para reforzar tu aprendizaje de las estructuras condicionales simple y doble, realiza en clase con la guía de tu profesor la siguiente actividad.

1. Retoma el algoritmo que diseñaste en *PSeInt* del problema de estructuras simple y doble en la actividad Ejercitando mis conocimientos pág. 41.

2. Codifica el algoritmo en *C++* en el editor *Code::Blocks*.

3. Comprime en una carpeta todos los archivos generados y guarda con el nombre compuesto por tus iniciales seguidas de **_PC_P3_E01** y comparte con tu profesor por el medio que indique.

3. Condicionales dobles anidadas - (if-else-if). Las selectivas dobles anidadas se usan para encadenar múltiples condiciones, es como una escalera. El programa revisa la primera condición; si es falsa, revisa la segunda, y así sucesivamente. En estas estructuras uno de los dos bloques se ejecutará sí o sí.

Algoritmo	Código en *C++*
Si (condición(es)) entonces *Instrucción(es)* *De lo contrario Si (condición(es)) entonces* *Instrucción(es)* *De lo contrario* *Instrucción(es)* *Fin del Si*	*if(condición(es)){* *instrucción(es);* *}else if(condición(es)){* *instrucción(es);* *}else{* *instrucción(es);* *}*

¿Sabías qué...?

Otro elemento que puede agregarse en un programa son comentarios que el desarrollador coloca y que el compilador ignora. Hay dos formas de agregar la nota:

1. // esto es una nota de una línea.
2. /*instrucción que lee dato*/.

Para saber más...

Observa el video Estructura condicional *If else if* en *C++* para ampliar la explicación de tu profesor. Accede a él escaneando el Código QR.

Ejemplo de selectiva doble anidada. Su problema y su algoritmo en *PSeInt* se encuentran en el recurso digital de la página 71.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int nivel = 0;
    char reto = ' ';
    cout << "Cuál es el nivel de jugador? ";
    cin >> nivel;
    cout << "¿Hace el reto diario? (S=si, N=no): ";
    cin >> reto;
    if ( nivel >= 20 ){
        if ( reto == 's' || reto == 'S' ){
            cout << "Skin Épica" << endl;
        } else {
            cout << "Skin Rara" << endl;
        }
    } else {
        if ( reto == 's' || reto == 'S' ){
            cout << "Caja Ítems" << endl;
        } else {
            cout << "Monedas x100" << endl;
        }
    }    return 0;
}
```

Ejercitando mis conocimientos

Para reforzar tu aprendizaje de las estructuras selectivas Anidadas, realiza en clase con la guía de tu profesor la siguiente actividad.

1. Retoma el algoritmo que diseñaste en *PSeInt* del problema de estructuras condicionales Anidadas en la actividad Ejercitando mis conocimientos pág. 44.

2. Codifica el algoritmo en *C++* en el editor *Code::Blocks*.

3. Comprime en una carpeta todos los archivos generados y guarda con el nombre compuesto por tus iniciales seguidas de **_PC_P3_E02** y comparte con tu profesor por el medio que indique.

3. Condicional Selector (switch-case). Es una alternativa al *if-else-if*, pero es más limpia cuando se quiere comparar el valor de una sola variable contra múltiples casos exactos. En esta estructura el ***'break;'*** es crucial, es el *else* del *switch*, se ejecuta si ningún caso coincide, pero si se olvida, el programa ejecutará ese caso y todos los que le siguen será un llamado *fall-through*.

Algoritmo	Código en *C++*
En caso de (op) *Caso op1:* *Instrucción(es)* *Interrumpir* *Caso op2:* *Instrucción(es)* *interrumpir* *Caso defecto:* *Instrucción(es)* *interrumpir* *Fin del caso*	*switch (op){* *case 1:* *instrucción(es);* *break;* *case 2:* *instrucción(es);* *break;* *default:* *instrucción(es);* *}*

Ejemplo de selectiva simple, su problema y algoritmo en *PSeInt* se encuentran en el recurso digital de la página 71.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int reaccion = 0;
    cout << "¿Qué reacción deseas pulsar? (1-5): ";
    cin >> reaccion;
    switch (reaccion) {
        case 1:
            cout << "Me gusta" << endl;
            break;
        case 2:
            cout << "Me encanta" << endl;
            break;
        case 3:
            cout << "Me divierte " << endl;
            break;
        case 4:
            cout << "Me asombra " << endl;
            break;
        case 5:
            cout << "Me entristece " << endl;
            break;
        default:
            cout << "Reacción no válida" << endl;
    }       return 0;
}
```

Para saber más...

Accede y observa el video Estructura condicional *Switch-case* en *C++* que profundiza en la explicación del tema. Hazlo escaneando el Código QR.

Estructuras repetitivas

Las también conocidas como bucle o ciclo, es una construcción que ejecuta un conjunto de instrucciones varias veces hasta que se cumple con una condición específica. Esto ayuda a automatizar tareas que se repiten, reduciendo la necesidad de escribir el mismo código una y otra vez.

Los ciclos más comunes son while, for y do-while.

1. Repetitiva Para (for). El bucle perfecto cuando se sabe exactamente cuántas veces se quiere repetir algo, a esto se le llama controlado por contador. Su sintaxis tiene 3 partes:

- Inicialización: se ejecuta una sola vez al empezar. Aquí se crea el contador, *int i = 0*
- Condición: se revisa antes de cada repetición. Si es *true*, el bucle se ejecuta, si es *false*, el bucle termina.
- Incremento: se ejecuta después de cada repetición. Aquí *i++* suma 1 a *i*.

Algoritmo	Código en C++
Para(inicio;condición(es);comportamiento) *Instrucción(es) 1* *Instrucción(es) n* *Fin Para*	*for(inicio;condición(es);comportamiento){* *instrucción(es) 1;* *instrucción(es) n;* *}*

Ejemplo de selectiva simple. Tanto el problema como el algoritmo en *PSeInt* se encuentran en el recurso digital de la página 71.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int dias=0,i=0;
    float likes=0,promedio=0;

    cout << "¿Cuántos días deseas evaluar? ";
    cin >> dias;
    for ( i=1 ; i <= dias ; i++ ){
     cout << "Porcentaje de likes día " << i << ": ";
     cin >> likes;
     promedio += likes;
    }
    promedio = promedio / dias;
    cout << "Promedio general de likes: " << fixed <<
setprecision(2) << promedio << endl;
    return 0;
}
```

Para saber más...

Observa el video Estructura repetitiva *For* en *C++* y conoce más de lo explicado en clase. Accede a él escaneando el Código QR.

Ejercitando mis conocimientos

Para reforzar tu aprendizaje de las estructuras repetitiva Para, realiza en clase con la guía de tu profesor la siguiente actividad.

1. Retoma el algoritmo que diseñaste en *PSeInt* del problema de estructura repetitiva Para, de la actividad Ejercitando mis conocimientos pág. 52.

2. Codifica el algoritmo en *C++* en el editor *Code::Blocks*.

3. Comprime en una carpeta todos los archivos generados y guarda con el nombre compuesto por tus iniciales seguidas de **_PC_P3_E03** y comparte con tu profesor por el medio que indique.

2. Repetitiva Mientras (while). Esta estructura sirve cuando no se sabe cuántas veces se repetirá, pero se sabe la condición que debe cumplirse para seguir ejecutándose. Es un bucle controlado por condición, es decir, que revisa la condición antes de entrar en él.

Algoritmo	Código en C++
Mientras (condición(es)) Haz *Instrucción(es)* *Fin Mientras*	*while(condición(es)){* *instrucción(es);* *}*

Ejemplo de estructura repetitiva mientras. El problema y el algoritmo en *PSeInt* se encuentran en el recurso digital de la página 71.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int dias=1,meta=0,pasosDia=0,acumulado=0;

    cout << "¿Cuál es tu meta de pasos? ";
    cin >> meta;

    while ( acumulado < meta ){
        cout << "Pasos dia " << dias << ": ";;
        cin >> pasosDia;
        acumulado += pasosDia;
        dias++;
    }

   cout << "¡Meta alcanzada en " << dias-1 << " días!" << endl;
    return 0;
}
```

Para saber más...

Observa el video Estructura repetitiva *While* en *C++* y conoce más de lo explicado en clase. Accede a él escaneando el Código QR.

Para saber más...

Accede y observa el video Estructura repetitiva *Do while* en *C++* y conoce más de su funcionamiento. Hazlo escaneando el Código QR.

3. Repetitiva Haz...Mientras (do-while). Esta estructura iterativa es casi igual al *while*, pero con una diferencia clave, la condición se revisa al final del bucle. Esto garantiza que el bloque de código se ejecutará al menos una vez.

Algoritmo	Código en C++
Haz *Instrucción(es)* *Mientras(condición(es))*	*do{* *instrucción(es);* *}while(condición(es));*

Ejemplo de estructura repetitiva Haz mientras. El problema y el algoritmo en *PSeInt* se encuentran en el recurso de la página 71.

```
#include <bits/stdc++.h>
 using namespace std;

 int main() {
     int NIP=0,NIP2=0;
     string nombre="";

     cout <<  "¿Cuál es tu nombre de usuario? ";
     cin >> nombre;
     cout << "Introduce el NIP secreto: ";
     cin >> NIP;
     cout << "Accede a UltraApp" << endl;

     do{
         cout << "NIP: ";
         cin >> NIP2;
     }while ( NIP != NIP2 );

     cout << "Acceso concedido. Bienvenid@ " << nombre
 << " a UltraApp"<< endl;
     return 0;
 }
```

Ejercitando mis conocimientos

Para reforzar tu aprendizaje de las estructuras repetitiva Haz mientras, realiza en clase con la guía de tu profesor la siguiente actividad.

1. Retoma el algoritmo que diseñaste en *PSeInt* del problema de estructura repetitiva Haz mientras, de la actividad Ejercitando mis conocimientos pág. 48.

2. Codifica el algoritmo en *C++* en el editor *Code::Blocks*.

3. Comprime en una carpeta todos los archivos generados y guarda con el nombre compuesto por tus iniciales seguidas de **_PC_P3_E04** y comparte con tu profesor por el medio que indique.

Concretando mis conocimientos

Con el objetivo de aplicar lo aprendido a lo largo de la progresión y demostrar cómo codificar instrucciones en lenguaje *C++* utilizando condicionales y bucles, de manera colaborativa resuelve la siguiente actividad:

1. Reúnete con tu equipo de trabajo.

2. Descarguen el archivo de indicaciones mediante el QR de al lado.
- **a.** Apliquen las fases del pensamiento computacional al problema planteado y organícenlas en un documento de *Word*. Redacten de manera clara cada paso que siguieron para darle solución.
- **b.** Codifiquen el algoritmo en lenguaje *C++* en el editor *Code::Blocks*.
- **c.** Compilen y ejecuten el programa.
- **d.** Verifiquen errores y en el caso de haberlos, corríjanlos.

3. Guarden en una misma carpeta los archivos:
- **a.** Documento de las fases del pensamiento computacional. Agreguen al final el nombre de los integrantes del equipo.
- **b.** Archivos de código y ejecutable generados.

4. Compriman la carpeta utilizando como nombre sus iniciales separadas por guion medio y seguidas del nombre **_PC1_P3_ CMC** y compartan con su profesor por el medio que indique.

Instrumento de evaluación

Revisa la siguiente lista de cotejo para que conozcas los criterios con los que tu profesor evaluará tu reporte escrito.

Indicador	Si	No	Puntos
Aplican correctamente las fases del pensamiento computacional			1
Utiliza las librerías adecuadas			1
Declara correctamente cada variable de acuerdo con el tipo de datos			1
Codifican las instrucciones en *C++*			2
Seleccionaron las estructuras de control adecuadas para resolver de manera óptima el problema			2
Utilizan los operadores aritméticos, lógicos y de relación de acuerdo con el algoritmo			1
No presenta errores de compilación			1
Resuelve de manera óptima el problema mediante lenguaje *C++*			1

Recurso digital

Escanea el QR para descargar el archivo con las indicaciones para realizar la actividad.

Demostrando mi aprendizaje

Para demostrar tu aprendizaje conceptual referente a los temas abordados en la Progresión 3, realiza la actividad interactiva, ingresa a ella escaneando el código QR.

Progresión 4

Programación estructurada en C++

Codifica en *C++* arreglos unidimensionales para almacenar, procesar y manipular conjunto de datos, determinando la ejecución de instrucciones de manera organizada y eficiente en la resolución de problemas.

Tiempo estimado: 6 horas

Tus metas serán:

- Identificar la utilidad de los arreglos unidimensionales en la resolución de problemas que requieren manejar múltiples valores del mismo tipos de dato.
- Representar soluciones a problemas cotidianos y académicos mediante el diseño de algoritmos que emplean arreglos unidimensionales.
- Codificar y ejecutar programas en *C++* que utilizan arreglos unidimensionales para almacenar, recorrer y procesar datos (suma, promedio, máximo, mínimo, búsqueda lineal y ordenamiento).

Recuperando lo que sabemos

Imagine que eres parte del equipo de desarrolladores de un nuevo videojuego llamada *"Pixel Quest: data Run"*. El sistema del juego sufrió una falla: los códigos de puntuación de los jugadores se perdieron, pero quedaron fragmentados dispersos que tú deberás reconstruir utilizando tus conocimientos de programación.

El sistema logró guardar el puntaje de 5 jugadores en variables separadas que harías para resolver lo siguiente:

1. ¿Cómo podrías calcular el promedio de los puntajes usando solo variables y operaciones básicas?

_____

_____

2. ¿Qué estructura repetitiva podrías usar para encontrar el puntaje más alto?

_____

_____

3. Si hubiera 100 jugadores, ¿qué problema tendrías al usar una variable para cada uno?

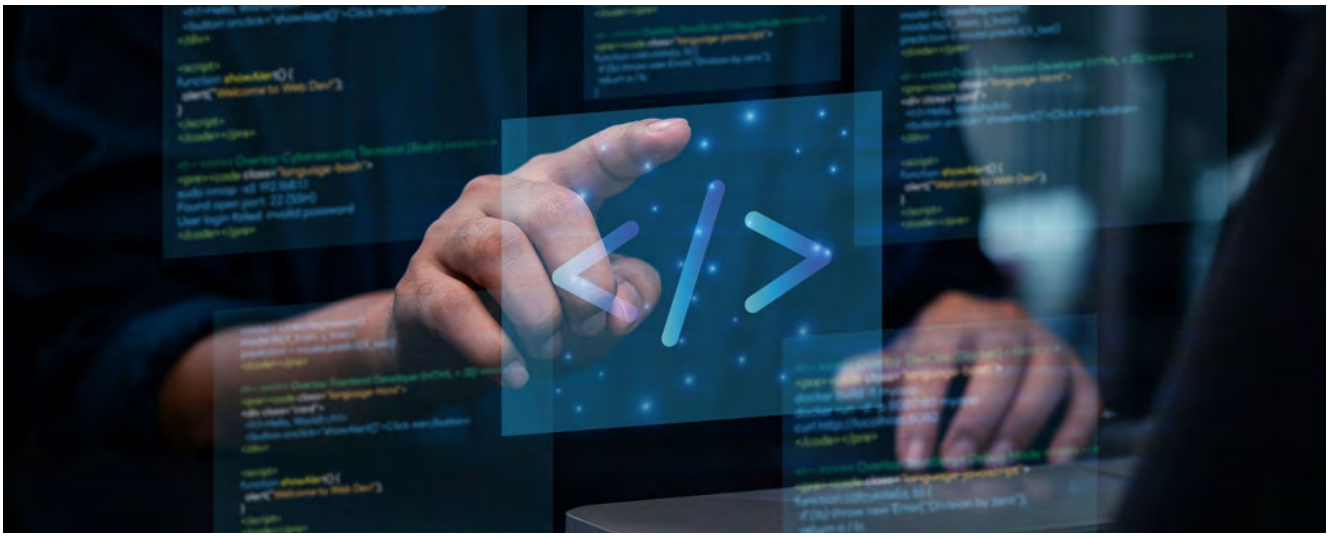
_____

_____

4. ¿Qué ventajas tendrías si todos estos valores estuvieran dentro de una sola estructura de datos?

_____

_____



Reactivando mis conocimientos

Cada vez que guardas objetos en un lugar específico, decides dónde colocarlos o acomodas algo siguiendo un orden, estás usando la misma lógica básica de los arreglos unidimensionales: una fila de elementos donde cada cosa ocupa una posición.

Imagina este escenario:

Al llegar a tu habitación, vacías tu mochila y colocas lo que traes en una repisa con compartimentos colocados en fila: libros, cuadernos, plumas, audífonos, llaves, etc. Cada objeto queda en un espacio diferente, uno junto al otro. Tu objetivo es tener tus cosas acomodadas de forma que puedas encontrarlas fácilmente cuando las necesites.

Para lograrlo, necesitas pensar en un procedimiento paso a paso: cómo decides qué espacio ocupará cada objeto, cómo buscas algo cuando lo necesitas y cómo reacomodas todo si quieres dejarlo ordenado.

1. En tu cuaderno o documento digital, escribe los pasos que seguirías para acomodar tus objetos en una repisa de una sola fila. Es decir, como decides en que orden pondrás los objetos, como reacomodas todos los objetos si quieres que queden ordenados por tamaño, importancia o frecuencia de uso.

2. Identifica los elementos del problema:

- **Datos de entrada:** ¿Qué cosas tienes que acomodar? (libros, llaves, audífonos, lápices, etc.), ¿Traes nuevos objetos que debas colocar?
- **Proceso:** ¿Qué reglas sigues para colocarlos?, ¿Los acomodas del más grande al más pequeño?, ¿Pones primero lo que usas más seguido?, ¿Buscas posición por posición cuando buscas un objeto?
- **Salida:** ¿Cómo queda tu repisa al final? . Una fila clara y ordenada donde cada cosa tiene una posición conocida y puedes acceder rápido a cualquier objeto.

3. Reflexiona y responde en tus notas:

- ¿Qué parte de tu procedimiento crees que un programa podría automatizar?
- ¿Cómo te ayudaría usar un entorno de desarrollo para simular tu "repisa" antes de programarla?
- ¿Qué ventajas tendría poder observar cómo se ejecutan tus pasos uno por uno en una simulación?

Comparte en clase tus pasos y reflexiones con tus compañeros y el profesor. Analicen juntos cuál de los procedimientos fue más claro, ordenado y eficiente, y comenten cómo ese mismo proceso podría transformarse en un algoritmo que use arreglos unidimensionales.

4.1 Estructuras de datos

Para saber más...

Escanea el código QR y observa el video Primer acercamiento a estructuras de datos.

¿Sabías qué...?

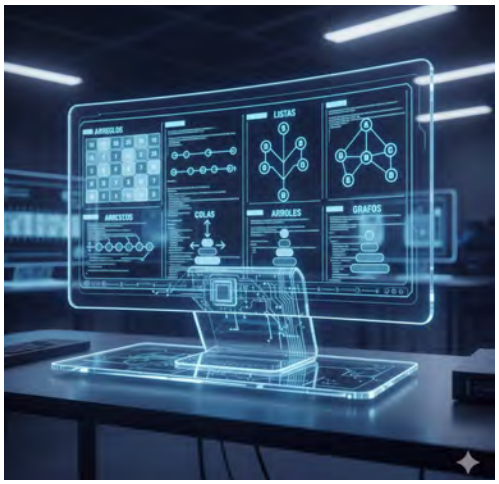
Una estructura de datos no solo almacena información, sino que también define cómo se accede, organiza y modifica.

Por ejemplo, una pila *(stack)* sigue el principio LIFO *(Last In, First Out)*, ideal para tareas como deshacer acciones en editores.

Las **estructuras de datos** constituyen un componente esencial en el ámbito de la informática y la programación, ya que permiten organizar, almacenar y gestionar la información de forma eficiente.

En términos generales, se entiende por **estructura de datos** una manera de recopilar datos y de definir relaciones entre ellos, así como los procedimientos para operar sobre esos datos.

Las estructuras de datos comprenden diversos tipos, entre los que se encuentran los **arreglos** *(arrays)*, las **listas enlazadas** *(linked lists)*, las **pilas** *(stacks)*, las **colas** *(queues)*, los **árboles** *(trees)*, los **grafos** *(graphs)*, las **tablas hash** *(hash tables)* y los **vectores**, entendidos como estructuras dinámicas.



Representación gráfica de estructuras de datos

Cada estructura de datos se adapta a determinados tipos de problemas en los que es necesario almacenar elementos, acceder a ellos, insertarlos, borrarlos o recorrerlos de forma eficiente.

Las estructuras de datos se emplean con el objetivo principal de organizar los datos contenidos dentro de la memoria de la computadora. Así, la experiencia con estructuras de datos comienza desde el momento que en los programas usan variables de tipos primitivos (*char, short, int, float, etc*).

El correcto uso de una **estructura de datos** adecuada contribuye a optimizar el rendimiento de programas, tanto en términos de tiempo (por ejemplo, tiempos de búsqueda, acceso o inserción) como de espacio (uso de memoria).

Por ejemplo, un arreglo permite acceso directo (aleatorio) a sus elementos con complejidad constante, lo cual lo hace útil cuando se conoce de antemano el número de elementos y se pretende un acceso rápido. Por tanto, es habitual que, antes de diseñar un algoritmo, se seleccione la estructura de datos más conveniente para la operación que se desea llevar a cabo.

Arreglos unidimensionales

En programación, un arreglo unidimensional, también conocido como vector o arreglos, es una estructura de datos que permite almacenar múltiples elementos del mismo tipo en ubicaciones contiguas de memoria. Esta característica facilita el acceso rápido a cualquier elemento mediante su índice, lo que permite operaciones eficientes como lectura, escritura y recorrido.

En lenguajes como *C++*, los arreglos son una herramienta fundamental para manejar grandes cantidades de datos. Son especialmente útiles cuando se necesita almacenar múltiples valores en una sola variable, como números, cadenas de texto o incluso tipos de datos más complejos. Una de las principales ventajas de los arreglos es que permiten organizar y acceder a la información de forma ordenada.

Por ejemplo, si se tiene la necesidad de crear un programa para registrar las calificaciones obtenidas por los estudiantes en un examen. Si se usaran variables individuales para cada alumno, el programa se volvería poco práctico, especialmente si el grupo es numeroso, ya que habría que declarar una variable para cada calificación. En cambio, utilizando un arreglo unidimensional, se pueden almacenar todas las calificaciones de manera más ordenada y eficiente, ya que cada posición del arreglo representaría la calificación de un estudiante diferente. De esta forma, el uso de arreglos facilita el manejo, consulta y procesamiento de múltiples datos del mismo tipo sin necesidad de declarar una variable por cada valor.

En *C++*, los arreglos tienen una limitación importante: su tamaño debe definirse al momento de la declaración. Esto significa que el tamaño será fijo durante la ejecución del programa aun y cuando no se necesiten todas las casillas de este. Existen otras estructuras de datos que si permiten que su tamaño sea aumentado o disminuido de manera dinámica según los elementos que se ingresen o se eliminen de ellos como, por ejemplo: colas, pilas, listas enlazadas, árboles, grados, etc.

▶ Declaración

En *C++* la sintaxis para declarar un arreglo unidimensional es el siguiente:

```
tipo_de_dato nombre_identificador [tamaño];
```

Interpretación:

- ***tipo_de_dato.*** Indica el tipo de dato que almacenará el arreglo. Puede ser int para números enteros, float para números decimales, char para caracteres, o incluso string para cadenas de texto.
- ***nombre_identificador.*** Es el nombre que tendrá el arreglo. Debe seguir las mismas reglas que las variables: comenzar con una letra o guion bajo, no usar espacios ni caracteres especiales, no ser una palabra reservada, y ser descriptivo.

Estudiando

Dedica un tiempo a la lectura de las páginas correspondientes a los temas de **Estructuras de datos**. Realizar esta tarea, te facilitará el aprendizaje y realizar las actividades que el profesor guiará en las siguientes sesiones. Apóyate en alguna estrategia de lectura que te ayude a mejorar la comprensión lectora.

Para saber más...

Escanea el código QR y observa el video Arreglos unidimensionales en *C++*.

¿Sabías qué...?

Las colas *(queues)* son estructuras de datos que siguen el principio FIFO *(First In, First Out)*.

Se usan en sistemas como la gestión de procesos en sistemas operativos o en la impresión de documentos.

¿Sabías qué...?

Tamaño Fijo en arreglos.
En muchos lenguajes de programación de tipado estático (como *C++* o *Java*), una vez que declaras un arreglo, su tamaño es fijo y no se puede cambiar dinámicamente durante la ejecución del programa.

Si necesitas más espacio, debes crear un arreglo completamente nuevo y copiar los datos.
.

Conceptos clave

Indexación. Proceso de acceder o modificar elementos individuales dentro de un arreglo (también llamado *array*) mediante un número índice que identifica su posición.

- **[tamaño].** Entre corchetes se especifica el número de elementos que contendrá el arreglo. Este valor debe ser un número entero positivo y es obligatorio al declarar arreglos estáticos.

Veamos un ejemplo de declaración de un arreglo.

```
float grupo[10];
```

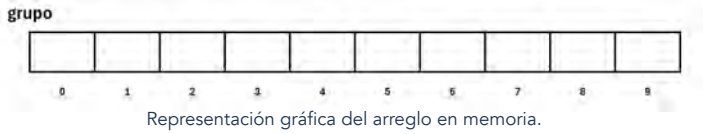
tipo de dato | nombre del arreglo | tamaño

Representación de la sintaxis de un arreglo unidimensional.

Interpretación:

- ***float.*** Indica el tipo de dato que almacenará el arreglo, en este caso números decimales.
- ***grupo.*** Es el nombre del arreglo, que nos permitirá identificarlo y acceder a sus elementos.
- ***[10].*** Representa el tamaño del arreglo, es decir, la cantidad de elementos que puede almacenar (en este caso, 10 valores de tipo float).

Al declarar un arreglo en *C++*, el programador está reservando un bloque de memoria contigua en la computadora. Este espacio se divide en posiciones, cada una destinada a almacenar un valor del tipo de dato especificado en la declaración.



Representación gráfica del arreglo en memoria.

En el lenguaje *C++*, los arreglos inician su indexación en 0 por razones relacionadas con la forma en que se calcula la posición de cada elemento en memoria. Es decir, el primer elemento estará en el índice 0 y el último será en el índice *n-1*. Donde **n** es el tamaño del arreglo.

Esta estructura permite realizar distintas operaciones, como:

- **Acceder** a un elemento específico mediante su índice.
- **Modificar** el valor almacenado en una posición.
- **Recorrer** el arreglo para aplicar cálculos o mostrar datos.

Para ilustrar de manera más precisa el funcionamiento de los arreglos y su aplicación en contextos de la vida cotidiana, puede considerarse el siguiente ejemplo: supóngase que una persona acude a un gimnasio que dispone de diez casilleros. En este caso, el primer casillero no se identifica con el número 1, sino con el 0, dado que dicho número representa la distancia desde el punto de inicio. Así, el casillero 0 se encuentra inmediatamente en la entrada (sin desplazamiento alguno), el casillero 1 está ubicado a un paso de distancia, el casillero 2 a dos pasos, y así sucesivamente. De este modo, el índice refleja el número de posiciones que es necesario avanzar desde el inicio para acceder al elemento correspondiente.

Inserción de datos

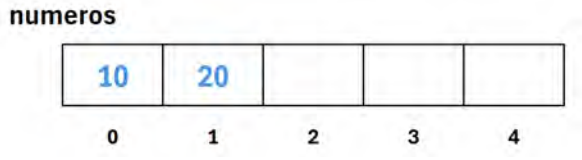
Cuando el arreglo ya está declarado, es posible asignar valores a cada posición. Estos valores deben coincidir con el tipo de dato definido en la declaración del arreglo.

Por ejemplo, si el arreglo fue declarado como *int*, únicamente puede almacenar números enteros; intentar guardar cadenas de texto en un arreglo de tipo numérico sería incorrecto y generaría errores.

La asignación de valores puede realizarse de varias maneras:

1. *Asignación individual:* se indica el índice exacto donde se almacena el dato.

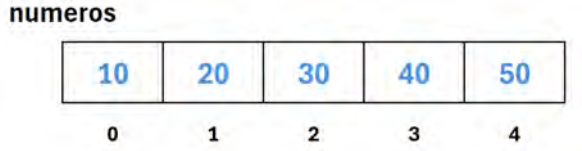
```
int numeros[5];
numeros[0] = 10;
numeros[1] = 20;
```



Representación gráfica del arreglo ***numeros*** en memoria por asignación individual.

2. *Inicialización directa:* se asignan los valores al momento de declarar el arreglo.

```
int numeros[5] = {10, 20, 30, 40, 50};
```



Representación gráfica del arreglo ***numeros*** en memoria por inicialización directa.

3. *Lectura mediante entrada de usuario:* es una de las formas más comunes para llenar un arreglo con datos, solicitando al usuario del programa que introduzca los valores desde el teclado usando un ciclo, el más indicado para hacerlo es la estructura for:

```
int calificaciones[5];
for (int i = 0 ; i < 5; i++ ) {
cout << "Ingrese la calificación " << i +1  << ": ";
     cin >> calificaciones[i];
}
```

4. *Inicialización implícita:* en el lenguaje *C++*, también es posible que la declaración de arreglos este acompañada de una lista de valores escritos entre llaves, sin indicar explícitamente el tamaño del arreglo. Esta forma de declaración también es correcta y se denomina inicialización implícita.

```
int numeros[] = {5, 10, 15, 20, 25};
```

¿Sabías qué...?

Acceso Directo.
La principal ventaja de un arreglo unidimensional es su capacidad de acceso directo (o aleatorio).

Esto significa que puedes acceder a cualquier elemento de la lista (por ejemplo, al elemento en la posición 50) en el mismo tiempo que tardarías en acceder al primer elemento, sin necesidad de recorrer los anteriores.

¿Sabías qué...?

Si se intenta acceder a una posición que no existe en el arreglo (por ejemplo, arreglo[10] cuando solo hay 5 elementos), el programa puede comportarse de manera inesperada.

A este error se le llama desbordamiento de índice *(index out of range)* y es una de las causas más comunes de fallos en programas que usan arreglos.

.

Este método resulta útil cuando se conoce el conjunto exacto de valores que se desea almacenar, ya que evita errores al calcular manualmente el tamaño y mejora la legibilidad del código. Sin embargo, es importante recordar que todos los valores deben coincidir con el tipo de dato declarado para el arreglo.

Acceso

El acceso a los elementos de un arreglo unidimensional en *C++* se realiza mediante el uso de índices colocados entre corchetes ([ ]). Tal como se ha mencionado anteriormente, los índices en *C++* comienzan en 0, lo que significa que el primer elemento del arreglo se encuentra en la posición cero. Para obtener el valor de un elemento específico, basta con escribir el nombre del arreglo seguido del índice correspondiente entre corchetes. Por ejemplo, si se tiene un arreglo de cinco elementos, el primer elemento se accede con el índice 0 y el último con el índice 4, ya que los índices válidos van desde 0 hasta *n-1*, donde n es el tamaño del arreglo.

Ejemplo:

```
int numeros[5] = {10, 20, 30, 40, 50};
cout << numeros[0]; // Imprime 10 (primer elemento)
cout << numeros[4]; // Imprime 50 (último elemento)
```

Es importante tomar en cuenta que intentar acceder a un índice fuera del rango definido (por ejemplo, ***numeros[5]***) provocará un error, ya que esa posición no existe en memoria. Este es uno de los errores más comunes que se pueden cometer al acceder a los datos de un arreglo.

En el siguiente ejemplo se muestra como un arreglo puede llenarse capturando sus datos a través de entrada de usuario y como imprimir todo su contenido una vez que fueron almacenado los datos en el.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    const int TAM = 5;            // Tamaño del arreglo
    string autobuses[TAM];        // Declaración del arreglo

    // Captura de datos
    cout << "Ingrese los nombres de " << TAM << " autobuses:" << endl;
    for (int i = 0; i < TAM; i++) {
        cout << "Autobus " << i + 1 << ": ";
        cin >> autobuses[i];
    }

    // Mostrar los datos capturados
    cout << "\nLista de autobuses ingresados:" << endl;
    for (int i = 0; i < TAM; i++) {
        cout << "Posicion [" << i << "] = " << autobuses[i]  << endl;
    }
 return 0;
}
```

Interpretación:

1. Se define una constante llamada ***TAM*** con valor ***5***, que indica el tamaño del arreglo. El uso de una constante permite que el tamaño sea fácil de modificar y evita errores al cambiarlo en varias partes del código.

2. Se declara un arreglo llamado ***autobuses*** de tipo ***string***, el cual podrá almacenar 5 nombres. Cada posición del arreglo corresponde a un índice que va desde ***0 hasta 4***

3. Se utiliza la estructura de control repetitiva. Ciclo ***for*** para recorrer las posiciones del arreglo y solicitar al usuario que ingrese los nombres de los autobuses.

- El ciclo inicia en ***i = 0*** y termina en ***i < TAM***, asegurando que se capturen exactamente 5 nombres.
- En cada iteración, se muestra un mensaje indicando el número del autobús y se almacena el valor ingresado en la posición correspondiente del arreglo.

4. Una vez ingresados los nombres, se utiliza otro ciclo ***for*** para recorrer el arreglo y mostrar cada elemento junto con su índice. Esto permite verificar que los datos fueron almacenados correctamente.

5. Finaliza el programa. Se utiliza el ***return 0***; para indicar que el programa terminó.

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza lo siguiente:

1. Inicia un nuevo programa de *C++* en *Code Block* que resuelva lo siguiente.

▶ Problema: Mi top de bandas legendarias del Rock

El rock nunca muere, solo se amplifica.

Imagina que eres parte de una comunidad estudiantil apasionada por el hard rock y el heavy metal clásico, y que están preparando un especial llamado "Las Leyendas del Rock".

Tu tarea es desarrollar un programa que permita registrar los nombres de las bandas legendarias favoritas de los estudiantes y luego mostrar una lista personalizada con un formato inspirado en un cartel de concierto.

Entrada

- La primera línea contiene un número entero *N*, que representa la cantidad de bandas que se van a registrar.
- Las siguientes *N* líneas contienen los nombres de las bandas (una cadena por línea).

Salida

El programa debe mostrar un mensaje con el siguiente formato:

_\,,/ Cartel Oficial: ¡Las Leyendas del Rock! _\,,/

¿Sabías qué...?

El ciclo *for* es el más utilizado para recorrer arreglos porque permite controlar con precisión cuántas veces se repite una instrucción.

Como el tamaño del arreglo se conoce desde el inicio, el for ejecuta las acciones exactamente n veces, una por cada elemento.

Por eso, es más claro y menos propenso a errores que otros ciclos como *while*.

.

Relaciónalo con...

En los programas que utilizan arreglos, las variables acumuladoras y contadoras resultan esenciales para procesar y analizar los datos almacenados.

El acumulador permite obtener resultados como sumas o promedios, mientras que el contador facilita saber cuántos elementos se han recorrido o cumplen una condición.

Sin ellas, sería mucho más difícil obtener información útil del arreglo, ya que no se tendría control sobre los valores ni sobre la cantidad de datos procesados.

.

a. [Nombre de la primera banda]
b. [Nombre de la segunda banda]
...
N. [Nombre de la última banda]

¡Prepárate para una noche de riffs y solos épicos!

Ejemplo:

Entrada	Salida
6 *AC/DC* *Metallica* *Iron Maiden* *Led Zeppelin* *Black Sabbath* *Guns N' Roses*	*_\,,/ Cartel Oficial: ¡Las Leyendas del Rock! _\,,/* *1. AC/DC* *2. Metallica* *3. Iron Maiden* *4. Led Zeppelin* *5. Black Sabbath* *6. Guns N' Rouses* *¡Prepárate para una noche de riffs y solos épicos!*

2. Una vez terminado el programa, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el programa en lenguaje *C++* creado en *Code Blocks* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P4_E01**.

4 Hazle llegar a tu profesor el algoritmo probado y listo para recibir retroalimentación.

▶ Operaciones

En programación, los arreglos unidimensionales permiten una vez creados, realizar distintas operaciones para recorrer, operaciones aritméticas con los elementos, buscar elementos dentro de ellos y ordenar los elementos; estas acciones son esenciales para resolver problemas de cálculo, organización y análisis de datos.

Recorrido (Traversal)

Recorrer un arreglo significa acceder a cada uno de sus elementos, normalmente usando un ciclo ***for***, esto permite leer, mostrar o modificar.

```
int arreglo[5] = {16,17,18,16,17};
for (int i = 0; i < n; i++) {
    cout << arreglo[i] << endl;
}
```

El ciclo for utiliza la variable i como índice que va aumentando en cada repetición, lo que permite pasar o visitar todas las posiciones del arreglo una por una.

Operaciones aritméticas con los elementos de un arreglo

Una de las ventajas del uso de arreglos en programación es que permiten realizar operaciones aritméticas de manera ordenada y sistemática sobre un conjunto de datos. Para poder realizar estas operaciones, es común recorrer el arreglo con un ciclo y aplicar la operación deseada con cada elemento.

```
int suma = 0;
int calificaciones [5] = {8,9,7,8,9};
for (int i = 0; i < n; i++) {
    suma += calificaciones[i];
}
cout << "La suma de las calificaciones es: " << suma;
```

De forma similar, también pueden realizarse otras operaciones, como calcular promedio, multiplicar los valores de una arreglo, entre otras.

Búsqueda (Search)

La operación de búsqueda de elementos en un arreglo es una de las operaciones más usuales en programas que los utilicen, ya que no solo basta con almacenar datos, sino que también es necesario localizar un valor especifico dentro del arreglo; para esto se pueden utilizar diferentes métodos de búsqueda, siendo el de búsqueda lineal el más simple.

La búsqueda lineal consiste en recorrer el arreglo desde el primer hasta el último elemento, comparando cada valor con el dato que se desea encontrar; si el valor coincide el programa muestra la posición y detiene el recorrido si se llega al final del arreglo sin encontrar coincidencias, significa que el valor no se encuentra en el arreglo.

La búsqueda lineal es muy útil en especial cuando los elementos del arreglo no están ordenados.

```
bool encontrado = false;
int buscado = 5;
int numeros [6] = {3,7,12,5,9,4};
for (int i = 0; i < 6 && !encontrado ; i++) {
    if (arreglo[i] == x) {
       cout << "El número " << buscado << " se encuentra en
la posición " << i + 1 << endl;
        encontrado = true;
    }
}

if ( !encontrado ) {
       cout << "El número no está en el arreglo. " << endl;
}
```

Para saber más...

Escanea el código QR para consultar infografía de la Jerarquía de operadores

.

Para saber más...

Escanea el código QR para observar e interactuar con una infografía interactiva sobre las operaciones con arreglos.

.

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza lo siguiente:

1. Inicia un nuevo programa de *C++* en *Code Blocks* que resuelva lo siguiente.

▶ **Problema: Buscando a tu personaje en el *Torneo Multiverso Gamer***

El *Torneo Multiverso Gamer* reúne a los personajes más icónicos de distintos videojuegos: desde héroes legendarios hasta villanos temidos.

Los organizadores del torneo te han pedido desarrollar un programa que permita buscar si un personaje específico fue seleccionado para competir este año.

Tu tarea es:

- Registrar los nombres de los personajes participantes.
- Permitir que el usuario escriba el nombre de un personaje a buscar.
- Indicar si ese personaje está inscrito o no fue seleccionado para el torneo.

Entrada

- Un número entero *N*, que representa la cantidad de personajes inscritos.
- Las siguientes *N* líneas contienen los nombres de los personajes (una cadena por línea).
- Finalmente, una cadena con el nombre del personaje a buscar.

Salida

El programa debe mostrar un mensaje con el siguiente formato:

- Si el personaje sí fue seleccionado:
- El personaje **[nombre]** está listo para la batalla en el *Torneo Multiverso Gamer :)*
- Si el personaje no fue seleccionado:

*El personaje **[nombre]** no participa en el Torneo Multiverso Gamer este año :(*

Entrada	Salida
¿Cuántos personajes se inscribieron? 8 Personaje 1: Mario Personaje 2: Link Personaje 3: Pikachu Personaje 4: Sonic Personaje 5: Kratos Personaje 6: Master Chief Personaje 7: Lara Croft Personaje 8: Scorpion ¿Cuál personaje quieres saber si participará en el *Torneo Multiverso Gamer*? Kratos	El personaje Kratos está listo para la batalla en el *Torneo Multiverso Gamer.*

2. Una vez terminado el programa, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique. Guarda el programa en lenguaje *C++* creado en *Code Blocks* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P4_E02**.

3. Hazle llegar a tu profesor el algoritmo probado y listo para recibir retroalimentación.

Además de recorrer o buscar datos específicos, es común que un programa que usa arreglos se necesite identificar el valor más grande o más pequeño de todos los elementos dentro del arreglo. Esto es útil para cuando se desea encontrar por ejemplo la calificación más alta o más baja en un conjunto de calificaciones grupales.

Para lograr esto, se utiliza un proceso muy sencillo:

1. Se declara una variable auxiliar que almacene temporalmente el primer valor del arreglo.

2. A medida que el ciclo recorre los elementos, se compara cada uno con el valor almacenado.

3. Si se encuentra un valor mayor (o menor), este reemplaza al anterior.

Encontrar el valor mínimo o máximo

Ejemplo de mínimo:

```
int minimo = arreglo[0];
int numeros[5] = {10,25,8,32,17};
for (int i = 1; i < n; i++) {
    if (arreglo[i] < minimo) {
        minimo = arreglo[i];
    }
}

cout << “El valor más pequeño es: ” << menor << endl;
```

Ejemplo de máximo:

```
int maximo = arreglo[0];
int numeros[5] = {10,25,8,32,17};
for (int i = 1; i < n; i++) {
    if (arreglo[i] > maximo) {
        maximo = arreglo[i];
    }
}

cout << “El valor más grande es: ” << mayor << endl;
```

¿Sabías qué...?

En *C++*, las variables de tipo *string* son, en realidad, arreglos de caracteres.

Cada letra ocupa una posición dentro del arreglo, lo que permite acceder, recorrer y modificar sus elementos igual que en cualquier otro arreglo.

Por ejemplo, nombre[0] representa el primer carácter de la cadena almacenada en la variable nombre.

.

Para saber más...

Escanea el código QR y observa e interactúa con la infografía interactiva sobre el método burbuja.

.

Ordenamiento (Sorting)

Otra de las acciones que comúnmente se realizan con los arreglos, es la de ordenar los elementos, esto significa organizarlos siguiendo un criterio, ya sea de manera ascendente o descendente. Uno de los algoritmos más sencillos para ordenar los elementos de un arreglo es el método burbuja *(Bubble sort)*, el cual basa su funcionamiento en comparar pares de elementos adyacentes e intercambiar si no están en el orden correcto, este proceso se repite varias veces hasta que todos los elementos quedan ordenados.

Algoritmo de ordenamiento (método de la burbuja):

```
for (int i = 0; i < n-1; i++) {
    for (int j = 0; j < n-i-1; j++) {
        if (arreglo[j] > arreglo[j+1]) {
            int temp = arreglo[j];
            arreglo[j] = arreglo[j+1];
            arreglo[j+1] = temp;
        }
    }
}
```

Estas operaciones son esenciales para el manejo de datos en programas más complejos. A continuación, se describen algunas de ellas con ejemplos prácticos en *C++*.

El lenguaje *C++* cuenta con la función ***sort()*** que ordena el arreglo de menor a mayor utilizando algoritmos avanzados optimizados internamente.

```
int N = 5;
int numeros[N];
for ( int i = 0 ; i < N ; i++ ){
     cin >> numeros [i];
}
sort (numeros, numeros + N);
```

Otra de las funciones que el lenguaje *C++* tiene para trabajar de manera eficiente estructuras de datos es la función ***reverse()*** esta permite invertir el orden de los elementos de un arreglo, logrando que al combinar la función ***sort()*** con la ***reverse()*** se obtenga el arreglo ordenado de manera descendente o viceversa según se necesite.

```
reverse (numeros, numeros + N);
```

Estas dos funciones, ***sort()*** y ***reverse()*** proporcionan herramientas eficientes para trabajar de una forma rápida y con las estructuras de datos, optimizando el rendimiento y reduciendo la complejidad del código.

En el siguiente ejemplo se muestra como un arreglo puede llenarse capturando sus datos a través de entrada de usuario y como imprimir todo su contenido una vez que fueron almacenado los datos en el. Además de hacer un ordenamiento **Ascendente** y **Descendente**.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    const int TAM = 5;           // Tamaño del arreglo
    string autobuses[TAM];       // Declaración del arreglo

    // Captura de datos
    cout << "Ingrese los nombres de " << TAM << " autobuses:" << endl;
    for (int i = 0; i < TAM; i++) {
        cout << "Autobus " << i + 1 << ": ";
        cin >> autobuses[i];
    }

    // Mostrar los datos capturados
    cout << "\nLista de autobuses ingresados:" << endl;
    for (int i = 0; i < TAM; i++) {
        cout << "Posicion [" << i << "] = " << autobuses[i] << endl;
    }

    // Ordenar en orden ascendente
    sort(autobuses, autobuses + TAM);
    cout << "\nLista ordenada en orden ASCENDENTE:" << endl;
    for (int i = 0; i < TAM; i++) {
        cout << autobuses[i] << " ";
    }
    cout << endl;

    // Ordenar en orden descendente usando reverse()
    reverse(autobuses, autobuses + TAM);
    cout << "\nLista ordenada en orden DESCENDENTE:" << endl;
    for (int i = 0; i < TAM; i++) {
        cout << autobuses[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza lo siguiente:

1. Inicia un nuevo programa de *C++* en *CodeBlocks* que resuelva lo siguiente.

▶ Problema: Ordenando los Likes del día

Un grupo de estudiantes de bachillerato está participando en un reto escolar en redes sociales donde cada uno publica un video corto.

Tu tarea es desarrollar un programa que registre el número de "likes" que obtuvo cada video durante el día y luego muestre esos valores ordenados de menor a mayor para conocer quién tuvo más interacción.

Entrada
Un número entero *N*, que representa la cantidad de videos publicados.

Una línea con *N* números enteros separados por espacio, donde cada número indica la cantidad de "likes" obtenidos por cada video.

Salida
El programa debe mostrar un mensaje con el siguiente formato:

Likes ordenados del reto escolar
[Like1] [Like2] [Like3]...[LikeN]

¡Así quedó el ranking del día!

Ejemplo:

Entrada	Salida
¿Cuántos videos se publicaron el día de hoy? 6 Likes: 320 Likes: 150 Likes: 475 Likes: 290 Likes: 510 Likes: 410	Likes ordenados del reto escolar 150 290 320 410 475 510 ¡Así quedó el ranking del día!

2. Una vez terminado el programa, ejecuta y prueba su funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el programa en lenguaje *C++* creado en *Code Blocks* colocando en el nombre del archivo tus iniciales seguidas de **_PC_P4_E03**.

4. Hazle llegar a tu profesor el algoritmo probado y listo para recibir retroalimentación.

Concretando mis conocimientos

Es tiempo de demostrar tu aprendizaje de los temas de Estructuras de datos, reúnete con tu equipo de trabajo y de manera colaborativa realicen lo siguiente:

1. Inicia un programa de *C++* en *Code Blocks* que resuelva el siguiente problema: ***Raking Gamer – Busca tu posición en la tabla!.***

Durante el evento ***"GameZone Challenge"***, cientos de jugadores compiten en línea por alcanzar la mayor puntuación posible.

Tú eres el encargado de desarrollar un programa que ayude a organizar el ranking de jugadores y buscar si un jugador específico logró entrar en el **Top del Día**.

Tu programa debe:

- Leer puntuaciones obtenidas por los jugadores.
- Ordenarlas de mayor a menor (para formar el ranking).
- Permitir buscar una puntuación específica y decir si aparece en el ranking o no. Además de la posición en la que está.

Entrada

- Un número *N*, que indica cuántos jugadores participaron.
- Una línea con *N* números enteros, separados por espacio, que representan las puntuaciones obtenidas.
- Una línea adicional con una puntuación *X* que se desea buscar.

Salida

El programa debe mostrar los siguientes mensajes:

- La lista ordenada de puntuaciones (de mayor a menor).
- Un mensaje indicando si la puntuación buscada se encuentra en el ranking y en que posición se encuentra.

Mensaje de salida en caso de encontrar la puntuación:
La puntuación *X* está en la posición *N* del ranking. ¡Buen trabajo gamer!.

Mensaje de salida en caso de no encontrar la puntuación:
La puntuación *X* no aparece en el ranking. ¡Sigue practicando!.

Entrada	Salida
¿Cuántos jugadores participaron? **8** Puntaje jugador 1: **540** Puntaje jugador 2: **720** Puntaje jugador 3: **610** Puntaje jugador 4: **450** Puntaje jugador 5: **900** Puntaje jugador 6: **660** Puntaje jugador 7: **830** Puntaje jugador 8: **700** ¿Cuál puntaje deseas buscar? **720**	Ranking del GameZone Challenge 1: 900 2: 830 3: 720 4: 700 5: 660 6: 610 7: 540 8: 450 La puntuación 720 está en la posición 3 del ranking. ¡Buen trabajo gamer!
¿Cuántos jugadores participaron? **5** Puntaje jugador 1: **300** Puntaje jugador 2: **250** Puntaje jugador 3: **400** Puntaje jugador 4: **380** Puntaje jugador 5: **500** ¿Cuál puntaje deseas buscar? **600**	Ranking del GameZone Challenge 1: 500 2: 400 3: 380 4: 300 5: 250 La puntuación 600 no aparece en el ranking. ¡Sigue practicando!

2. Una vez terminado el algoritmo ejecuta y comprueba su correcto funcionamiento con los casos de ejemplo y otros valores adicionales que tu profesor indique.

3. Guarda el programa de *C++* creado en *Code Blocks* colocando en el nombre del archivo tus iniciales seguidas de **PC_P4_CMC**.

4. Comparte con tu profesor por el medio que indique tu algoritmo probado y listo para recibir evaluación.

Instrumento de evaluación

Revisa la siguiente lista de cotejo para que conozcas los criterios con los que tu profesor evaluará tu programa en *Code Blocks*.

Indicador	Si	No	Puntos
El algoritmo solicita correctamente los datos de entrada: número de jugadores, puntuaciones y puntuación a buscar.			1
Se muestra un mensaje inicial indicando el propósito del programa (organizar ranking y buscar puntuación).			1
Se ordenan correctamente las puntuaciones de mayor a menor.			3
Se busca correctamente la puntuación deseada y se determina si está en el ranking.			2
Se muestra la posición de la puntuación buscada con un mensaje claro y motivador.			2
Se muestra un mensaje adecuado si la puntuación no se encuentra en el ranking.			1

Demostrando mi aprendizaje

Para demostrar tu aprendizaje conceptual referente a los temas abordados en esta progresión, realiza la actividad interactiva, ingresa a ella escaneando el código QR.

Valorando mi aprendizaje

La evaluación es un proceso continuo de formación, útil para recabar evidencias sobre el logro de los aprendizajes, con oportunidad de retroalimentación y mejora de los resultados.

En este apartado se presentan algunas actividades e instrumentos, que te guían en la valoración de los aprendizajes que adquiriste progresivamente en las secuencias didácticas anteriores. Responde honestamente a cada una de ellas.

Reflexionando lo que aprendí

Contesta las siguientes preguntas y reflexiona sobre tu desempeño en estas dos progresiones.

- ¿Qué fue lo más importante que aprendiste sobre las estructuras de control en *C++* y cómo consideras que este conocimiento te ayuda a pensar de manera más lógica en otras materias?

- Reflexiona sobre la importancia de las estructuras de control en la creación de programas funcionales. ¿Qué relación encuentras entre organizar instrucciones en *C++* y organizar tareas o actividades en tu vida diaria?

- Piensa en un programa que hayas codificado utilizando arreglos en *C++*. ¿Qué fue lo más difícil y qué aprendiste del proceso de prueba y error?

- Después de todo lo aprendido, ¿qué consideras que necesitas practicar más en programación estructurada en *C++* y por qué crees que eso será importante para tu avance académico?

Actividad alternativa

Resuelve para reforzar tu aprendizaje e incrementar tu evaluación.

1. Crea un video donde se observe y expliques:
   - **a.** El diseño de un programa en lenguaje C++ para calcular el promedio de calificaciones.
   - **b.** El resultado debe mostrar el promedio de un alumno y los mensaje de nivel según el resultado:
     - "Excelente" para calificación 9-10
     - "Suficiente" para calificación 7-8
     - "Necesitas apoyo para calificación 5-6".
   - **c.** Emplea el tipo de estructura que optimice la operación.

2. Envía el video a tu profesor para que evalúe tu desempeño.

Autoevaluación

La autoevaluación es un mecanismo de autocontrol que te ayuda a regular tu aprendizaje. Marca con una √ la columna que corresponda a tu nivel de dominio en los aspectos de aprendizaje en cada meta.

Metas	Criterios	Nivel de dominio		
		Sí lo logro	En proceso	Aún no lo logro
Distingue la sintaxis básica de *C++* y la utilidad de las estructuras de control para organizar la ejecución de instrucciones.	Identifico la sintaxis básica de *C++* (declaración de variables, operadores, entradas y salidas).			
	Reconozco la utilidad de las estructuras de control selectivo y repetitivo.			
Representa soluciones a problemas cotidianos y académicos mediante algoritmos que incorporan estructuras de control secuenciales y repetitivas.	Selecciono la estructura de control adecuada según el problema planteado.			
	Explico en qué casos conviene utilizar cada tipo de estructura de control.			
Codifica, compila y ejecuta programas en *C++* validando su funcionamiento y corrigiendo errores en el uso de estructuras de control.	Traduzco un algoritmo a código *C++* utilizando correctamente las estructuras de control.			
	Compilo y ejecuto programas verificando que funcionen según lo esperado.			
Identifica la utilidad de los arreglos unidimensionales en la resolución de problemas que requieren manejar múltiples valores del mismo tipo de datos.	Reconozco problemas que requieren almacenar muchos valores del mismo tipo.			
	Identifico cuándo es más eficiente usar un arreglo en lugar de variables aisladas.			
Representa soluciones a problemas cotidianos y académicos mediante el diseño de algoritmos que emplean arreglos unidimensionales.	Diseño algoritmos que incluyen la declaración, lectura y uso de arreglos unidimensionales.			
	Estructuro los pasos para recorrer un arreglo (búsqueda, conteo, acumulación, etc.).			
Codifica y ejecuta programas en *C++* que utilizan arreglos unidimensionales para almacenar, recorrer y procesar datos (suma, promedio, máximo, mínimo, búsqueda lineal y ordenamiento).	Declaro y utilizo correctamente arreglos unidimensionales en *C++*.			
	Implemento operaciones de procesamiento: suma, promedio, máximo, mínimo.			
	Realizo búsqueda lineal en un arreglo.			
Lo mejor que aprendí fue:				
Lo que necesito reforzar es:				

Calificación que doy a mi desempeño:	Excelente	Satisfactorio	En desarrollo	Inicial

Coevaluación

Evalúa el desempeño general de tu equipo de trabajo durante el desarrollo de las actividades de aprendizaje colaborativas. Coloca el valor correspondiente en la columna Evaluación y suma para conocer el resultado del trabajo por equipo.

Buen trabajo (3)	Algo nos faltó (2)	Debemos mejorar (1)	Evaluación
Organizamos el trabajo estipulando tareas, prioridades y plazos.	Se organizó el trabajo, pero no se estipularon tareas, prioridades o el plazo de entrega final.	No hubo organización para realizar nuestros trabajos.	
Cumplimos cada uno con las tareas asignadas en el plazo estipulado.	Casi todos los miembros del equipo cumplimos con las tareas asignadas y el plazo estipulado; teniendo que resolver lo que a otros les fue encomendado.	Un solo miembro del equipo realizó todos los productos.	
Todos participamos activamente en la elaboración de los productos.	Casi todos los miembros del equipo participamos activamente en la elaboración de los productos.	No hubo participación de los miembros del equipo en la elaboración de los productos.	
La calidad de los productos que elaboramos fue la adecuada para su entrega.	La calidad de los productos que elaboramos fue en su mayoría la adecuada para su entrega.	No se cumplió con la calidad adecuada de los productos para su entrega.	
		Total	___ de 12

Progresión 5

Robótica Educativa

Simula sistemas robóticos mediante aplicaciones gráficas y la programación en Arduino, implementando estructuras básicas de código (*setup y loop*), funciones elementales (*pinMode, digitalWrite, delay*), control de salidas múltiples mediante ciclos y retardos, así como el uso de sensores y actuadores para resolver problemas simples de automatización.

Tiempo estimado: 12 horas

Tus metas serán:

- Identificar la importancia de la robótica educativa como herramienta para comprender la interacción entre *hardware* y *software*.
- Configurar sistemas básicos de automatización en un entorno gráfico controlando las salidas.
- Codificar programas con estructuras de control, funciones básicas, bucles, sensores y actuadores en simuladores.

Recuperando lo que sabemos

Este cuestionario es de recuperación de conocimientos previos, es útil para identificar tus saberes y habilidades y cómo los relacionas con la realidad, además te ayudará a comprender mejor los temas de esta progresión. No es necesario que conozcas los términos técnicos; lo importante es expresar cómo entiendes o aplicarías cada situación, haz tu mejor esfuerzo y detecta aquellos aspectos que no conoces o dominas para enfocar tu estudio.

1. Cuando escuchas las palabras "Robótica Educativa", ¿qué es lo primero que te viene a la mente? ¿Qué esperas aprender o ser capaz de construir utilizando robots en un entorno de aprendizaje?

_____

_____

_____

2. Si tuvieras que explicarle a un amigo qué es un robot, ¿qué dirías que son sus tres partes esenciales para que pueda realizar una tarea? (*Pista: Piensa en cómo interactúa con el mundo, cómo "piensa" y cómo se mueve*).

_____

_____

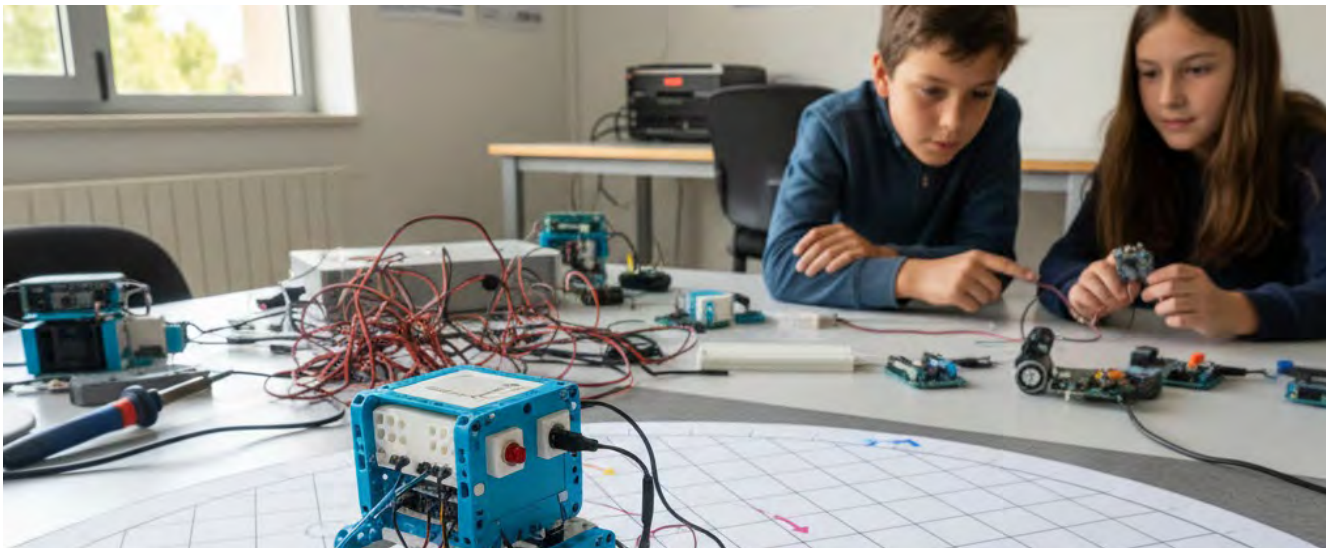
_____

3. ¿De qué manera crees que el conocimiento de la programación y la robótica podría ser relevante o útil para tu futuro, incluso si no planeas ser ingeniero o programador?

_____

_____

_____



Reactivando mis conocimientos

Cada vez que diseñas un robot, decides qué sensores usar o programas sus movimientos, estás aplicando lógica, secuencia y análisis de problemas, los mismos principios que se utilizan al crear un algoritmo.

Imagina este escenario:

En tu clase de robótica, tu equipo debe programar un robot para que recoja objetos de diferentes colores y los coloque en cajas según su categoría. Tu objetivo es diseñar un procedimiento paso a paso que permita al robot identificar el color del objeto, decidir a qué caja llevarlo y completar la tarea de manera eficiente.

1. En tu cuaderno o documento digital, escribe los pasos que seguirías para que un robot detecte un objeto más cercano, leer el color del objeto, decidir la caja correspondiente al color, mover el robot hacia la caja correcta, colocar el objeto en la caja y repetir el proceso hasta que no queden objetos.

2. Identifica los elementos del problema:

- **Datos de entrada:** ¿Qué información recibes?
- **Proceso:** ¿Qué acciones o reglas aplicas para clasificar colores?
- **Salida:** ¿Cuál es el resultado final o estado ideal de tu pantalla?

3. Reflexiona y responde en tus notas:

- ¿Qué parte de tu procedimiento crees que un programa podría automatizar?
- ¿Cómo te ayudaría usar un entorno de desarrollo para simular tu algoritmo antes de programarlo?
- ¿Qué ventajas tendría poder observar cómo se ejecutan tus pasos uno por uno en una simulación?

Comparte en clase tus pasos y reflexiones con tus compañeros y el profesor. Analicen juntos cuál de los procedimientos fue más claro, ordenado y eficiente, y comenten cómo ese mismo proceso podría transformarse en un algoritmo computacional para el robot.

5.1 Introducción a la robótica

Conceptos clave

Robot. Fue inventada por el escritor checo *Karel Capek* para designar a los autómatas de su obra teatral de ciencia ficción *R.U.R* (Robots Universales Rossum), estrenada en Praga en 1921. Una palabra acuñada por *Capek* a partir del término checo **robota**, que hace referencia al trabajo duro.

¿Sabías qué...?

La robótica no solo consiste en construir máquinas con forma humana. En realidad, es la combinación de mecánica, electrónica y programación para crear sistemas capaces de realizar tareas autónomas.

¡Mucho más que robots humanoides de película!

Historia

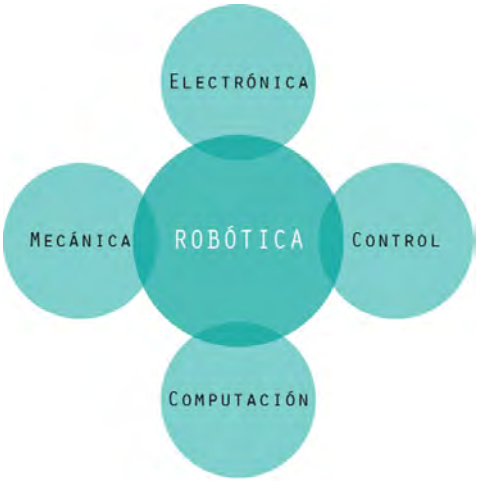
En la actualidad, la tecnología forma parte esencial de la vida diaria desde los teléfonos inteligentes hasta los sistemas automatizados en fábricas, la innovación tecnológica está presente en casi todo lo que nos rodea.

Dentro de este contexto surge la robótica, una disciplina que despierta gran interés y curiosidad. Sin embargo, cuando se pregunta qué es la robótica, lo más común es relacionarla únicamente con robots humanoides o con escenas de películas de ciencia ficción; esta idea, aunque popular, no refleja la verdadera esencia de la robótica.

Las maquinas a las que comúnmente se les llaman robots, son diseñadas para actividades muy diversas: desde ensamblar piezas en una fábrica hasta explorar otros planetas. Más allá de la imagen futurista que se suele tener, la robótica es una herramienta práctica que busca facilitar procesos, mejorar la calidad de vida y ampliar las posibilidades de la ciencia y la tecnología.

¿Qué es la Robótica y Qué es un Robot?

La robótica es una rama interdisciplinaria que combina conocimientos de ingeniería mecánica, electrónica, eléctrica, control y ciencias de la computación.



Ramas de la ciencia que integran la robótica.

Su objetivo es diseñar, fabricar y programar máquinas automáticas con cierto grado de inteligencia, capaces de ejecutar tareas específicas. Por su parte, un robot es una máquina programable que puede tomar decisiones basadas en la estructura de su programa y realizar acciones de manera automática.

En términos sencillos, la robótica se dedica al diseño y desarrollo de robots capaces de sustituir o complementar actividades humanas en distintos ámbitos, como

la industria, el hogar o los entornos científicos; esta disciplina no se limita únicamente a especialistas, sino que también resulta accesible para estudiantes y entusiastas interesados en aprender y experimentar.

Clasificación de la Robótica

La robótica se aplica en diversos campos, entre los que destacan:

- **Robótica industrial:** diseña robots para procesos de manufactura y ensamble, como la producción automotriz, la clasificación de piezas y el empaquetado de alimentos. Su objetivo es reducir costos, optimizar tiempos y minimizar errores humanos.
- **Robótica de servicio:** incluye robots que brindan asistencia a las personas, como sistemas quirúrgicos, robots de limpieza, dispositivos para entretenimiento, exploración y rescate.
- **Robótica espacial:** se enfoca en la creación de robots para la exploración del espacio. Ejemplos son los robots *Spirit* y *Opportunity*, enviados a Marte para investigar la posible existencia de agua.

Robot Spirit que forma parte del Programa de Exploración de Marte de la **NASA**. Enviada en enero de 2004.

Robótica Educativa y STEM

Además de sus aplicaciones industriales y científicas, la robótica tiene un papel fundamental en la educación. La **robótica educativa**, también llamada robótica pedagógica, busca que los estudiantes se familiaricen con la programación y el diseño de robots desde edades tempranas.

Esta disciplina se adapta al nivel académico del alumno, ofreciendo herramientas sencillas para la educación básica y sistemas más complejos para niveles superiores.

La robótica educativa forma parte del modelo **STEM** (*Science, Technology, Engineering and Mathematics*), que promueve el aprendizaje práctico de la ciencia, la tecnología y las matemáticas.

Este enfoque permite desarrollar habilidades cognitivas, pensamiento lógico y creatividad, mientras se aprende a resolver problemas mediante la experimentación.

Conceptos clave

STEM. El modelo educativo ha evolucionado hacia STEAM, incorporando el arte como un elemento fundamental para fomentar la creatividad y la innovación en los procesos de aprendizaje.

¿Sabías qué...?

STEAM en educación no tiene nada que ver con videojuegos. Es un enfoque que une ciencia, tecnología, ingeniería, arte y matemáticas para desarrollar habilidades del siglo XXI.

¡Muy distinto a Steam, la tienda digital de juegos!

Tipo de educación con orígenes en la década de los 90.

¿Sabías qué...?

La electricidad es simplemente el movimiento de cargas eléctricas, pero es la base de casi todo lo que usamos a diario: desde tu celular hasta el refrigerador.

¡Un fenómeno invisible… pero imprescindible!

Conceptos clave

Oscilante. Describe algo que oscila, es decir, que se mueve o varía de forma repetitiva y recurrente alrededor de una posición de equilibrio

Relaciónalo con...

La Corriente Directa (CD) es la que entregan las pilas, baterías y paneles solares. Fluye en un solo sentido, lo que la hace ideal para electrónica y dispositivos portátiles.

¡La energía perfecta para lo que se lleva en el bolsillo!

Conceptos básicos de electricidad y electrónica

La electricidad y la electrónica forman los cimientos de casi toda la tecnología moderna, y comprenderlas es indispensable para adentrarse en el mundo de la robótica. La **electricidad** se define como el movimiento de cargas eléctricas a través de un material conductor, lo que permite transportar energía y activar dispositivos como motores, luces o sensores; por otro lado la **electrónica**, es la rama de la tecnología que estudia, controla y aprovecha ese flujo de electricidad mediante componentes específicos como resistencias, diodos, transistores, microcontroladores y circuitos integrados.

La electricidad entonces se origina por el movimiento de partículas cargadas llamadas **electrones**, que poseen carga **negativa**. Cuando estos electrones se desplazan por un conductor metálico, generan lo que se conocer como **corriente eléctrica.**

Un ejemplo cotidiano es el cargador de un teléfono celular, al conectarlo a la corriente, la electricidad fluye por el cable para alimentar el dispositivo y permitir que la batería se recargue.

Para describir y medir la electricidad se emplean unidades específicas:

- **Voltio (V):** representa la fuerza que impulsa a los electrones a moverse por un circuito.
- **Amperio (A):** indica la cantidad de corriente eléctrica que está circulando en un conductor.

Existen dos formas principales de corriente eléctrica:

- **Corriente alterna (CA) o AC (en inglés):** es el tipo de corriente eléctrica en la que el sentido del flujo de los electrones (la polaridad) cambia periódicamente, a intervalos regulares, con un patrón oscilante que se representa como una onda sinusoidal. Debido a esta variación de polaridad, la corriente alterna fluye en ambos sentidos dentro del circuito.

La velocidad a la que ocurre esta oscilación se mide en hertzios (Hz), que indican cuántos ciclos se producen por segundo. En muchos países europeos y de otras regiones, la frecuencia es de 50 Hz, mientras que en países como México y Estados Unidos es de 60 Hz.

La corriente alterna es ideal para transportar energía a grandes distancias, ya que puede transformarse fácilmente a diferentes voltajes mediante transformadores. Por este motivo, es el tipo de corriente que se emplea en la distribución eléctrica desde las centrales generadoras hasta zonas urbanas y residenciales, es decir, es la que llega a los hogares y alimenta la mayoría de los electrodomésticos.

- **Corriente continua (CC) o DC (en inglés):** en la corriente continua, los electrones se desplazan de manera constante en una sola dirección, sin cambios de polaridad. Este flujo estable resulta especialmente adecuado para dispositivos electrónicos que requieren un suministro preciso y uniforme.

La corriente continua proviene de fuentes como pilas, baterías, paneles solares y otros generadores. Es la energía que almacenan las baterías de los teléfonos móviles, computadoras portátiles, controles remotos y muchos dispositivos portátiles. Incluso los aparatos que funcionan conectados a la corriente alterna suelen convertirla internamente en corriente continua para su funcionamiento interno.

Representación de las corrientes CC y AC.

Resistencia: La resistencia es la oposición que presenta un material al paso de la corriente eléctrica. Se mide en **ohmios (Ω)**.

En las resistencias hay códigos de colores. Estos códigos se utilizan para medir la "resistencia de las resistencias", por lo cual es fundamental conocer esta codificación.

El código de resistencia está compuesto por un número de bandas que oscilan entre 3 a 6. Siendo la más común la de 4 bandas.

Representación gráfica de la resistencia y su código de colores.

En la resistencia de la imagen podemos observar que:

- Banda 1 es de color azul por lo tanto corresponde a un 6.
- Banda 2 es de color rojo por lo tanto corresponde a un 2.
- Banda 3 es de color verde por lo tanto corresponde a multiplicarlo por 100,000 Ω.
- Banda 4 es de color dorado por lo tanto corresponde a una tolerancia del 5%

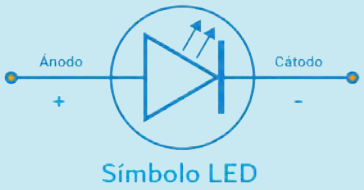
Entonces para calcular el valor de la resistencia sería:
62 x 100,000Ω = 6,200,000Ω

Por lo tanto, la resistencia tiene un valor de 6.2 megaohmios (MΩ) con una tolerancia de ± 5%.

¿Sabías qué...?

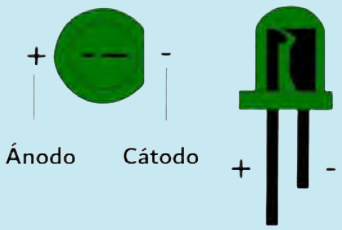
Las resistencias no solo controlan la corriente eléctrica: también vienen en distintos tamaños y potencias según lo que pueden soportar. Las más comunes son las pequeñas de ¼ de watt, ideales para proyectos con Arduino, pero también existen resistencias más grandes que disipan mayor calor.

¡Su tamaño no es al azar, sino una pista de cuánta energía pueden manejar!



Símbolo LED

Representación gráfica de la polaridad de un LED.



Representación gráfica de la polaridad de un LED.

Por ejemplo, si se conecta directamente un LED a una pila de 9 voltios sin utilizar resistencia, la corriente que circula será demasiado alta y el LED se quemará casi de inmediato. Esto ocurre porque el componente no está diseñado para soportar ese nivel de corriente. En cambio, al colocar una resistencia de **330 Ω** en serie con el LED, la corriente se reduce a un nivel seguro, evitando que el LED sufra daños, ya que la resistencia actúa como un limitador que controla cuánta corriente pasa por el circuito.

La Ley de Ohm establece la relación entre el voltaje (V), la corriente eléctrica (I) y la resistencia (R). Esta ley indica que, si se conocen dos de estos valores, es posible calcular el tercero mediante la fórmula: $V = I \times R$

Suponiendo que se cuenta con una pila de 9 voltios y una resistencia de 330 Ω conectada en un circuito sencillo, para calcular la corriente usando la ley de Ohm y sustituyendo los valores se obtiene:

$$I = \frac{V}{R} = \frac{9V}{330\Omega} \approx 0.027\,A\ (27\,mA)\ V/R$$

Este cálculo permite determinar la cantidad de corriente que pasará por el circuito y verificar si es segura para componentes sensibles, como un LED. De esta manera, la Ley de Ohm se convierte en una herramienta fundamental para diseñar y evaluar circuitos eléctricos básicos.

La **potencia eléctrica** indica la cantidad de energía que un dispositivo consume en un tiempo determinado. Se mide en vatios (W) y se calcula mediante la fórmula:

$$P=V\times I$$

Por ejemplo, si una lámpara está conectada a una fuente de 120 V y consume 0.5 A de corriente, su potencia puede calcularse sustituyendo estos valores en la fórmula:

$$P=V\times I$$
$$P=120V\times 0.5\,A$$
$$P=60\,W$$

Esto significa que la lámpara consume 60 vatios de potencia, dato que permite estimar su gasto energético y comparar la eficiencia entre distintos dispositivos.

La **polaridad** se refiere a la identificación del polo positivo (+) y negativo (–) en componentes eléctricos y electrónicos. Muchos dispositivos funcionan de manera correcta únicamente si están conectados con la polaridad adecuada.

Al conectar un LED por ejemplo, es necesario unir el ánodo (+) al lado positivo de la pila y el cátodo (–) al lado negativo. Si la polaridad se invierte, el LED no encenderá, ya que los diodos permiten el paso de la corriente únicamente en una dirección.

Respetar la polaridad es fundamental para asegurar el funcionamiento adecuado de los componentes electrónicos y evitar fallas en el circuito.

Aplicaciones

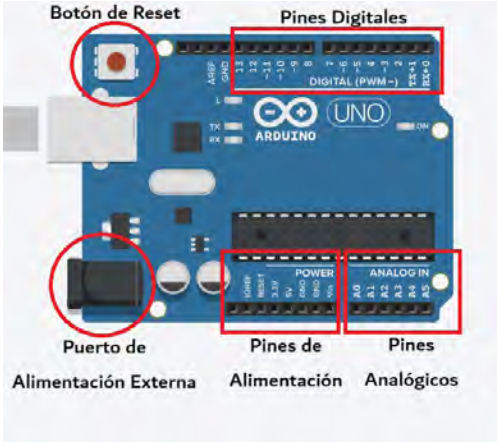
La **robótica educativa** se ha convertido en una herramienta clave para desarrollar habilidades tecnológicas en estudiantes, fomentando el aprendizaje práctico y la creatividad. Dentro de este contexto, Arduino destaca como una de las plataformas más accesibles y versátiles para introducir a los alumnos en el mundo de la programación y la electrónica aplicada.

Arduino es una placa de desarrollo basada en un microcontrolador que permite controlar dispositivos electrónicos mediante instrucciones programadas. Su diseño abierto y su facilidad de uso la convierten en el punto de partida ideal para proyectos de robótica, automatización y sistemas interactivos.

En la placa de **Arduino**, se pueden aplicar conceptos básicos de electricidad y electrónica, como voltaje, corriente, resistencia y polaridad en proyectos reales, reforzando la comprensión teórica con experiencias prácticas.

Dentro del proyecto **Arduino** existe una amplia variedad de placas de desarrollo diseñadas para diferentes tipos de proyectos. Algunas son modelos básicos, ideales para tareas sencillas con pocas entradas y salidas; otras son más avanzadas, ofreciendo mayor cantidad de pines y capacidades adicionales. También se encuentran placas con conexión a redes cableadas o inalámbricas, versiones compactas para proyectos donde el espacio es limitado e incluso placas flexibles pensadas para integrarse en prendas de vestir.

El modelo **Arduino UNO R3** ha sido seleccionado por su facilidad de uso, versatilidad y amplia compatibilidad con componentes y simuladores, lo que lo convierte en la opción ideal para introducir conceptos de electrónica, programación y robótica educativa.



Partes de la placa Arduino UNO R3.

¿Sabías qué...?

Arduino nació en 2005 en Ivrea, Italia, como un proyecto para facilitar la enseñanza de la electrónica y la programación. Fue creado por un equipo encabezado por *Massimo Banzi*, junto con *David Cuartielles, Tom Igoe, Gianluca Martino* y *David Mellis*.

¡Comenzó como una herramienta para estudiantes… y terminó revolucionando el mundo del hardware abierto!

Placa Arduino UNO R3

Relaciónalo con...

Existen diferentes placas **Arduino**, cada una diseñada para un tipo de proyecto. La **Arduino Uno** es la más famosa y perfecta para empezar; la **Mega** ofrece más pines para proyectos grandes; la **Nano** es ideal para espacios pequeños; y la **Leonardo** puede funcionar como teclado o ratón.

¡Cada placa tiene su propia personalidad y propósito en el mundo!

¿Sabías qué...?

Algunos pines digitales del **Arduino Uno** tienen la función especial PWM, marcada con el símbolo "~".

Esto permite crear señales que parecen analógicas, como variar el brillo de un LED o la velocidad de un motor.

¡No son analógicos reales, pero engañan muy bien!

Relaciónalo con...

Wokwi es un simulador de **Arduino** que funciona 100% online y sin límites: puedes usar **ESP32**, **Arduino Mega**, sensores avanzados y hasta pantallas OLED.

¡Ideal para probar proyectos complejos sin hardware real!

Partes de la placa Arduino UNO R3

- **Puerto de alimentación externa:** permite suministrar energía a la placa cuando no está conectada mediante el cable USB tipo B. Este puerto se utiliza en proyectos donde el Arduino debe funcionar de forma autónoma, sin depender de una computadora. Generalmente se conectan fuentes de voltaje externas, siempre dentro del rango permitido por la placa, que va de 7 V a 12 V. Superar este límite puede ocasionar daños irreversibles en el dispositivo.
    - Pines de alimentación:
    - Vin: voltaje de entrada si se emplea una fuente externa dentro del
    - rango de 7-12V.
    - 5V: suministra un voltaje regulado de 5 voltios, ideal para módulos y sensores compatibles con esta tensión.
    - 3.3v: ofrece una salida 3.3 voltios para componentes de baja tensión.
    - GND: representa la tierra del circuito, es fundamental para completar cualquier conexión eléctrica.
- **Pines analógicos (A0 a A5):** permiten leer señales analógicas provenientes de sensores como potenciómetros, sensores de temperatura o humedad; convierten la señal analógica en un valor digital que va de 0 a 1023, debido a una resolución interna de 10 bits.
- **Pines digitales (D0 a D13):** pueden configurarse tanto como entradas (INPUT) como salidas (OUTPUT). Algunos de estos tienen funciones especiales:
    - D0 y D1: se utilizan para comunicación serial (RT y TX).
    - D3, D5, D6, D9, D10 y D11: generan señales PWM (modulación por ancho de pulso) se activan con (*analogWrite*).
- **Botón de reset:** reinicia el programa cargado sin necesidad de desconectar el Arduino. Es útil para volver a ejecutar el código desde el principio.

Simuladores

Los simuladores son herramientas fundamentales para el aprendizaje práctico, ya que permiten experimentar y desarrollar habilidades sin necesidad de utilizar equipos físicos costosos o complejos. En el ámbito educativo, ayudan a comprender conceptos abstractos y favorecen la creatividad mediante una práctica segura y accesible. Antes de adquirir una placa Arduino y dedicar tiempo al montaje real, es recomendable validar el diseño del circuito de manera virtual. Los simuladores de Arduino permiten recrear la placa y sus componentes, probar el funcionamiento del circuito y verificar el código, asegurando que todo opere correctamente antes de llevarlo a la práctica física. Algunos de los simuladores más comunes son:

- **Tinkercad:** Plataforma intuitiva para diseño 3D y simulación de circuitos electrónicos. Ideal para principiantes y proyectos educativos.
- **Wokwi:** Simulador avanzado para microcontroladores como Arduino, ESP32 y Raspberry Pi. Perfecto para desarrolladores que buscan probar código y hardware virtualmente.
- **Proteus:** Software profesional para simulación de circuitos y sistemas embebidos. Muy usado en entornos académicos y de ingeniería.
- **Xevro:** Ofrece simuladores y herramientas para proyectos electrónicos y automatización, con enfoque en hardware real y entornos industriales.
- **IDE de Arduino:** Aunque es principalmente un entorno de desarrollo, incluye funciones de simulación y pruebas básicas para código antes de cargarlo en hardware real.

5.2 Aplicación TinkerCAD

Tinkercad es una plataforma en línea gratuita, desarrollada por Autodesk, que permite crear modelos 3D, simular circuitos electrónicos y programar microcontroladores mediante una interfaz intuitiva y accesible para usuarios de cualquier nivel. Por su diseño sencillo y visual, se ha convertido en una herramienta fundamental dentro de la educación tecnológica, ya que facilita el aprendizaje práctico sin necesidad de contar con equipos físicos costosos o complejos.

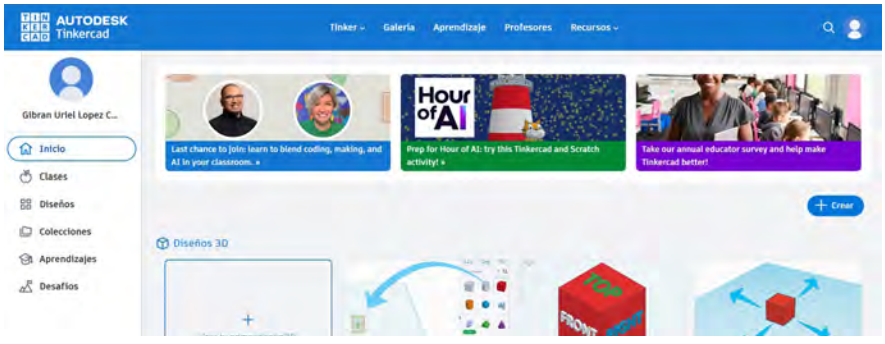
En el ámbito de la robótica educativa, ***Tinkercad*** aporta un espacio seguro y dinámico para experimentar con componentes electrónicos, comprender el funcionamiento de sensores y actuadores, y validar el comportamiento de un circuito antes de construirlo en la realidad. Su simulador de circuitos permite observar en tiempo real cómo interactúan las conexiones, el voltaje, la corriente y las instrucciones de programación, lo que ayuda a relacionar conceptos teóricos con resultados concretos.

Para comenzar a usar la plataforma es necesario crear una cuenta:

1. Accede al sitio web oficial.
   - 1.1 En cualquier navegador escribe ***https://www.tinkercad.com.***
2. Haz clic en "Join Now" o "Únete ahora".
   - 2.2 El botón se encuentra en la parte superior derecha de la página.
3. Selecciona el tipo de cuenta.
   - 3.1 Personal: Para uso individual.
4. Elige el método de registro.
   - 4.1 Puedes registrarte usando:
     - Correo electrónico y contraseña (creando una cuenta nueva), o Cuenta de *Google*, *Apple* o *Microsoft* (inicio rápido).
5. Completa la información requerida.
   - Nombre de usuario, correo electrónico y contraseña.
6. Acepta los términos y condiciones.
   - Marca la casilla correspondiente.
7. Verifica tu correo electrónico.

Interfaz gráfica

Al ingresar con la cuenta creada a *Tinkercad* se muestra la pantalla principal de la aplicación.



Pantalla de inicio en Tinkercad.

Para saber más...

Escanea el código QR y observa el video *Tinkercad*.

¿Sabías qué...?

Tinkercad permite simular circuitos electrónicos y programar un Arduino como si fuera un dispositivo real, sin necesidad de tener componentes físicos.

Gracias a su simulador, puedes probar sensores, LEDs, motores y hasta escribir código en Arduino *C++* para ver cómo funciona al instante. Esto convierte a ***Tinkercad*** en una de las herramientas más accesibles y seguras para aprender electrónica y robótica desde cualquier computadora.

Relaciónalo con...

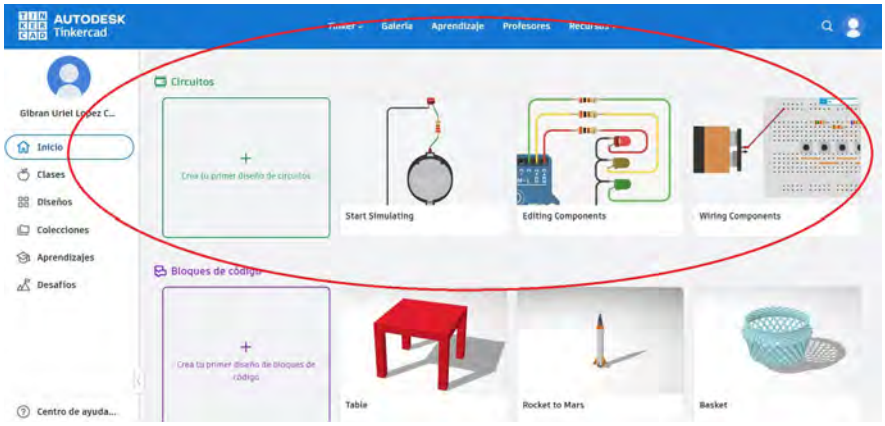
En la interfaz de ***Tinkercad***, el área central es tu mesa de trabajo, donde colocas componentes, cables y tu **Arduino**. Es como un banco de pruebas digital listo para experimentar. ¡Ahí es donde tus ideas toman forma!

¿Sabías qué...?

En la barra derecha de ***Tinkercad***, encontrarás la biblioteca de componentes: LEDs, botones, motores, resistencias, sensores y más.

Solo arrastras y sueltas lo que necesitas.

Para ingresar al apartado de Circuitos, una vez en la pantalla principal es necesario desplazarse hacia abajo donde se muestran distintas secciones como: Diseños 3D, Circuitos y Bloques de código.



Pantalla de inicio en Tinkercad en la sección de Circuitos.

En esta sección es donde se pueden crear diseños de circuitos con **Arduino** y acceder a la simulación de su funcionamiento. Al seleccionar la opción Crea tu primer diseño de circuitos, se muestra la interfaz con las herramientas para construir y programar circuitos.

Interfaz en la creación de circuitos.

Panel de acciones: permite realizar diversas operaciones de edición, como copiar, pegar y eliminar elementos, además de deshacer o rehacer cambios. También permite modificar el color de las líneas y seleccionar el tipo de conexión, con opciones como Normal, Conexión, Cocodrilo y Automático. Adicionalmente, ofrece la posibilidad de rotar los componentes para ajustar su orientación.

Panel Simulador/Programación: muestra el código fuente correspondiente al circuito y permite ejecutar su simulación para verificar el funcionamiento. Además, ofrece la opción de compartir el proyecto con otras personas de manera sencilla.

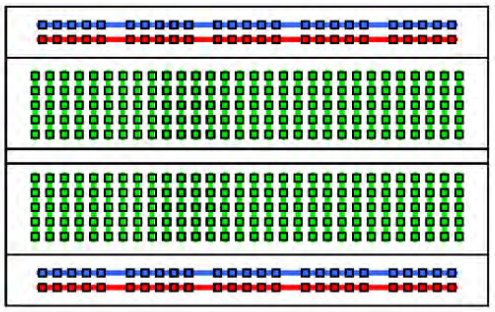
Área de trabajo: espacio donde se colocarán y organizarán todos los componentes necesarios para construir los circuitos que se diseñarán posteriormente.

Categorías / Componentes: se visualiza inicialmente la categoría Básicos, donde los componentes visibles pertenecen a dicha sección. El panel ofrece dos opciones principales: Básicos y Todos, además de un espacio dedicado a elementos específicos como placas Arduino, dispositivos *Micro:bit* y ensamblajes de circuitos.

Componentes básicos

Protoboard

También llamado *breadboard* constituye una plataforma de pruebas utilizada en el ámbito del diseño electrónico para la construcción rápida de circuitos eléctricos temporales sin necesidad de soldadura. Su estructura modular se compone de una matriz de perforaciones interconectadas internamente, lo que facilita la inserción y conexión segura de componentes electrónicos de manera sencilla y eficiente.



Conexiones internas de un *Protoboard*.

La *protoboard* se organiza en distintas secciones que facilitan el montaje de circuitos:

- **Zona verde:** corresponde a los orificios que se encuentran interconectados internamente y que permiten insertar componentes tanto en el montaje físico como en el simulador de *Tinkercad*.
- **Riel azul:** forma parte de los rieles de alimentación. Todos los orificios de este riel están conectados entre sí y se utilizan para distribuir la **alimentación positiva (+)** a los diferentes puntos del circuito.
- **Riel rojo:** también perteneciente a los rieles de alimentación, cuenta con orificios interconectados entre sí y se emplea para establecer la **alimentación negativa (–)** o la conexión a tierra del circuito.

Componente	Descripción
 **Resistencia**	• Función: Limita el flujo de corriente en el circuito. • Uso común: Protección de LEDs y control de corriente en diferentes componentes.
 **LED**	• Función: Emite luz cuando circula corriente en la dirección correcta. • Uso común: Indicadores visuales en proyectos electrónicos.

Relaciónalo con...

La *protoboard* recibe el nombre de *breadboard* porque, en sus inicios, los ingenieros utilizaban literalmente tablas de cortar pan para montar sus primeros circuitos. Durante las décadas de 1950 y 1960, estas tablas servían como base para sujetar componentes y realizar pruebas de manera provisional.

¿Sabías qué...?

El primer LED visible fue creado en 1962 por *Nick Holonyak Jr.*, quien trabajaba en General Electric.

Su luz era roja y muy tenue, pero marcó el inicio de la iluminación moderna.

¡Hoy los LEDs están por todas partes, desde pantallas hasta semáforos!

Componente	Descripción
 **Potenciómetro**	• Función: Resistencia variable que permite ajustar el voltaje en un circuito. • Uso común: Control de brillo, volumen o velocidad en motores.
 **Switch**	• Función: Abre o cierra el circuito para controlar el flujo de corriente. • Uso común: Encendido y apagado de dispositivos.
 **Diodo**	• Función: Permite el paso de corriente en una sola dirección. • Uso común: Protección contra polaridad inversa.
 **Condensador**	• Función: Almacena energía eléctrica temporalmente. • Uso común: Filtrado de señales y estabilización de voltaje. • Nota: También es llamado capacitor no polarizado.
 **Condensador polarizado**	• Función: Almacena energía eléctrica temporalmente. Solo que este condensador tiene una terminal en positivo (+) y una negativa (-), y se debe conectar respetando la polaridad. Caso contrario podría dañarlo. • Uso común: Filtrado de señales y estabilización de voltaje. • Nota: También es llamado capacitor polarizado.
 **Batería 9V**	• Función: Proporciona energía eléctrica al circuito, permitiendo que los componentes del circuito simulado funcionen. • Uso común: Alimentar circuitos simples con LED's y resistencias.

¿Sabías qué...?

El primer condensador fue creado en 1745 y se llamó **Botella de Leyden,** un curioso frasco capaz de almacenar carga eléctrica.

Hoy los capacitores están en cada fuente de poder, filtro y circuito electrónico.

¡Un invento del siglo XVIII que aún sostiene la tecnología del siglo XXI!

Componentes de entrada

Componente	Descripción
 **Pulsador**	• Función: Controlar el flujo eléctrico de manera temporal. • Uso común: Activar funciones específicas en proyectos, como encender un led o iniciar una acción.
 **Sensor de distancia (Ultrasónico)**	• Función: Enviar pulsos ultrasónicos y calcular el tiempo que tarda en regresar para determinar la distancia. • Uso común: En la detección de obstáculos.
 **Teclado (Keypad 4x4)**	• Función: Envía señales al microcontrolador de Arduino según la tecla presionada. • Uso común: Sistemas de seguridad, ingreso de contraseñas y control de menús en proyectos electrónicos.
 **Conmutador SPST**	• Función: Abrir o cerrar el circuito eléctrico. • Uso común: Control básico de alimentación de circuitos simples. Por su siglas en inglés significa (*Single Pole Single Throw*).
 **Sensor IR**	• Función: Detectar la presencia de objetos, medir distancia o recibir señales mediante luz infrarroja. Esto se logra emitiendo un haz de luz IR y analizando la reflexión o interrupción de ese haz. • Uso común: Detección de obstáculos, control remoto y sistemas de seguridad.
 **Sensor de temperatura**	• Función: Detecta variaciones térmicas y envía datos al microcontrolador. • Uso común: Control de climatización, monitoreo ambiental y proyectos de IoT.
 **Sensor de humedad**	• Función: Mide el nivel de humedad presente en la tierra o sustrato, detectando conductividad entre sus dos sondas. • Uso común: Sistemas de riego automatizados y monitoreo de humedad en macetas o cultivos.

¿Sabías qué...?

Los **sensores ultrasónicos** se basan en tecnologías desarrolladas durante la Segunda Guerra Mundial para sonar y radar.

Décadas después, se miniaturizaron hasta convertirse en los pequeños módulos que usan robots y **Arduino**.

¡De tecnología militar... a ojos electrónicos para tus proyectos!

¿Sabías qué...?

Los **primeros sensores de temperatura** modernos se basan en principios descubiertos en el siglo XIX, cuando científicos como **Seebeck** y **Thomson** estudiaron cómo los metales cambiaban con el calor.

¡Un principio físico antiguo convertido en tecnología esencial para tus proyectos!

Para saber más...

Escanea el código QR e interactúa con la infografía Componentes de conexión en *Tinkercad*.

Componente	Descripción
 **Sensor de gas**	• Función: Convertir la concentración de gas en una señal eléctrica que puede ser interpretada por microcontrolador como Arduino. • Uso común: Sistemas de seguridad para detectar fugas de gas, alarmas domésticas y proyectos IoT para monitoreo ambiental.
 **Sensor PIR**	• Función: Detectar movimiento mediante la variación de radiación infrarroja emitida por objetos (como personas). • Uso común: Alarmas, sistemas de iluminación automática y seguridad.
 **Foto resistencia (LDR)**	• Función: Medir los niveles de iluminación. • Uso común: Encendido automático de luces, proyectos de ahorro energético y sensores de luz.

Estudiando

De forma individual elabora una **tabla comparativa** que te permita organizar, contrastar y comprender los temas de esta primera secuencia de la progresión 5.

1. En un documento digital crea una tabla comparativa incluyendo los siguientes ejes:

- Electricidad (concepto, unidades, tipos de corriente)
- Electrónica (definición, componentes básicos)
- *Protoboard* (estructura y función)
- *Tinkercad* (características y utilidades)
- Arduino (pines, alimentación y funciones)
- Conexiones eléctricas (normal, jumper, cocodrilo, automática)

2. Organiza la información asegurándote que muestre: definiciones esenciales, principales funciones, diferencias y similitudes, etc.

3. Guarda el documento con el nombre compuesto por tus iniciales seguidas de **_PC_P5_E01**.

4. Hazle llegar el documento a tu profesor por el medio que acuerden.

5.3 Programación en Arduino

Conceptos básicos

Para trabajar con Arduino es indispensable comprender las funciones fundamentales que intervienen en la programación de proyectos. La estructura de un programa y el uso adecuado de sus instrucciones básicas constituyen la base del desarrollo tanto en el entorno de simulación de *Tinkercad* como en una placa física.

Los proyectos en Arduino se organizan en dos bloques principales de código: *void setup()* y *void loop()*.

La función *void setup()* se ejecuta una sola vez al iniciar el programa y se utiliza para configurar los pines y establecer parámetros iniciales, como definir si un pin funcionará como entrada o salida. En contraste, la función *void loop()* contiene las instrucciones que deben repetirse de manera continua durante el funcionamiento del dispositivo, permitiendo que el programa esté en ejecución constante y responda a los cambios en el entorno.

Además de esta estructura básica, Arduino incorpora una serie de funciones esenciales para interactuar con los distintos componentes electrónicos:

- **pinMode():** establece el modo de operación de un pin, definiéndolo como entrada (**INPUT**) o salida (**OUTPUT**). Una configuración adecuada asegura que sensores, botones, LEDs u otros componentes funcionen correctamente.

- **digitalWrite():** envía un valor digital a un pin configurado como salida. Puede establecer un estado **HIGH** (encendido o nivel lógico alto) o **LOW** (apagado o nivel lógico bajo). Es decir, al verificar el estado de un pin digital, un valor **HIGH** corresponde lógicamente al número **1**, mientras que un valor **LOW** equivale al número **0**.

- **digitalRead():** lee el estado de un pin configurado como entrada y detecta si recibe un valor digital **HIGH** o **LOW**. Es decir, al verificar el estado de un pin digital, un valor **HIGH** corresponde lógicamente al número **1**, mientras que un valor **LOW** equivale al número **0**.

- **analogWrite():** envía una señal analógica simulada mediante modulación por ancho de pulso (**PWM**), lo que permite, por ejemplo, variar la intensidad de un LED o la velocidad de un motor. Los valores que puede recibir son del **0** al **1023**.

- **analogRead():** lee valores analógicos provenientes de sensores, devolviendo un número que representa la intensidad de la señal recibida. Los valores que puede recibir son del **0** al **1023**.

- **delay():** pausa la ejecución del programa durante un tiempo específico expresado en milisegundos. Durante esta pausa, **Arduino** detiene temporalmente todas sus operaciones antes de continuar. Es útil para generar intervalos visibles entre acciones o controlar la duración de ciertos procesos. Por ejemplo: *delay(2000)*, provoca que el programa se detenga por 2 segundos, y después seguirá ejecutando la siguiente instrucción.

Para saber más...

Escanea el código QR y observa el video Mi primer sistema simulado en *Arduino*.

Para saber más...

Escanea el código QR y observa el video Estructura de código *setup()* y *loop()* en *Arduino*.

Para saber más...

Escanea el código QR y observa el video Función de *pinMode(), digitalWrite(), delay()* en *Arduino*.

También es fundamental comprender el uso de los pines de Arduino, ya que cada uno cumple funciones específicas dentro del circuito. Los pines están divididos en dos categorías principales: salidas digitales, se encargan de controlar el encendido y apagado de algún componente; y las salidas análogas que sirven para controlar la intensidad como el brillo de un LED o la intensidad de un motor.

Pines de Arduino Uno R3

Conocer estas funciones facilita la correcta conexión de los componentes en la protoboard y asegura un uso adecuado tanto en el simulador de *Tinkercad* como en el montaje físico del proyecto.

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza lo siguiente:

1. Inicia un nuevo circuito en *Tinkercad* para crear un sistema simulado de encendido de un LED.
2. En el área de trabajo de *Tinkercad*, coloca una placa **Arduino UNO R3**.
3. Agrega un *protoboard* al área de trabajo.
4. Coloca un LED al *protoboard*, verificando la polaridad del LED. Recuerda que ánodo es positivo y cátodo es negativo.
5. Coloca una resistencia y ajusta al valor de ella en 220Ω.
6. Realiza la conexión del ánodo del LED a un pin digital de Arduino, usa el pin 13.
7. Realiza la conexión del cátodo del LED al pin de tierra GND de Arduino.
8. Clic en el editor de código en *Tinkercad* y en la estructura de *void setup()* definiremos que estamos utilizando el pin 13. Para ello escribimos el código *pinMode(13, OUTPUT)*. Con esto estamos definiendo que el pin 13 será utilizado como un pin de salida.
9. En la estructura *void loop()* agregaremos primeramente las siguientes líneas de código. *digitalWrite(13,HIGH)* y posteriormente la línea *delay(2000)*.
10. En la misma estructura *void loop()*, agrega las siguientes líneas de código. *digitalWrite(13, LOW)* y posteriormente la línea *delay(2000)*.
11. Una vez concluida la construcción del circuito, ejecuta la simulación y verifica que su funcionamiento sea el adecuado.
12. Exporta el circuito en formato de imagen y en un documento inserta la imagen y el código creado en *Tinkercad*.
13. Coloca el link del proyecto de *Tinkercad* en una sección del documento.
14. Guarda el documento colocando en el nombre del archivo tus iniciales seguidas de **_PC_P5_E02**.
15. Hazle llegar a tu profesor el documento por el medio que acuerden para recibir retroalimentación.

Para saber más...

Escanea el código QR y observa el video Control de salidas.

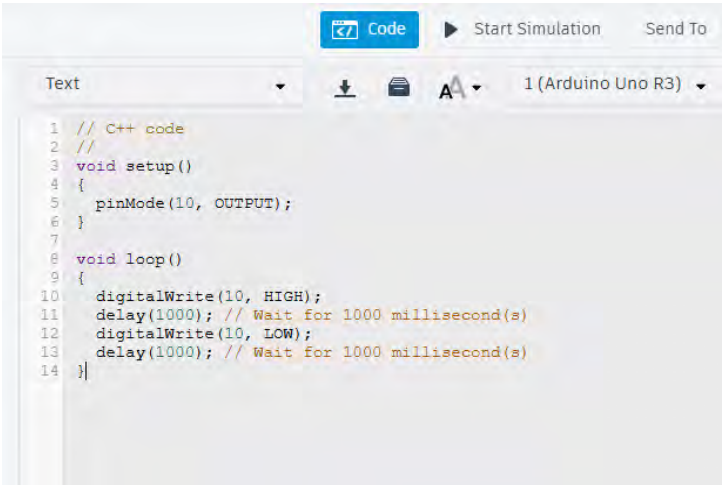
Programación básica en Arduino

La programación básica en Arduino se fundamenta en el uso de instrucciones y estructuras derivadas del lenguaje *C++*, lo que permite escribir código claro, ordenado y eficiente para controlar diversos componentes electrónicos. Este enfoque facilita que se apliquen habilidades de programación que no solo son útiles en Arduino, sino también transferibles a proyectos más avanzados en otros entornos.

El código en Arduino sigue una estructura estándar compuesta por dos bloques principales. El primero es *setup()*, donde se configuran los pines y parámetros iniciales, y el segundo es *loop()*, donde se ejecutan de manera repetitiva las instrucciones mientras la placa permanece encendida. Esta disposición refleja la lógica característica de *C++*, donde cada función cumple un propósito específico y debe escribirse siguiendo las reglas de sintaxis correspondientes, como el uso correcto de llaves, punto y coma, mayúsculas y minúsculas.

Uno de los conceptos fundamentales en la programación con Arduino es el control de salidas, que permite enviar señales eléctricas desde la placa hacia distintos dispositivos, como LEDs, zumbadores, motores o relés. Para ello, los pines deben declararse como OUTPUT dentro de *setup()* mediante la instrucción *pinMode()*, y luego activarse o desactivarse en el *loop()* utilizando funciones como *digitalWrite()*. La manera en que estas instrucciones se escriben y organizan sigue directamente la sintaxis de *C++*, exigiendo precisión en cada línea del programa.

Cuando se trabaja con salidas múltiples, el propósito es coordinar acciones simultáneas o secuenciales entre varios componentes. Esto implica identificar correctamente cada pin, establecer su función en el programa y estructurar el código en un orden lógico que permita ejecutar patrones, tiempos y combinaciones específicas. Gracias a la flexibilidad del lenguaje de programación basado en *C++*, es posible crear secuencias complejas, diseñar patrones luminosos, controlar varios actuadores a la vez o gestionar sistemas más amplios como semáforos, alarmas automatizadas o paneles indicadores.



Área de codificación de Arduino en *Tinkercad*.

¿Sabías qué...?

La programación en Arduino se basa en el enfoque modular de *C++*, donde el código se organiza en funciones que cumplen tareas específicas. Gracias a esta estructura, puedes dividir un proyecto en partes más pequeñas —como controlar luces, leer sensores o mover un motor— y hacer que Arduino las ejecute de forma ordenada.

Esta modularidad facilita entender, depurar y ampliar tus programas, tal como se trabaja en proyectos profesionales de *C++*.

Para saber más...

Escanea el código QR y observa el video Lectura en *Arduino*.

Dominar estas bases no solo permite comprender cómo interactúan *software* y *hardware*, sino que también fortalece el pensamiento lógico, la resolución de problemas y la capacidad para abstraer y modelar procesos. A medida que el estudiante observe cómo cada instrucción escrita en *C++* modifica el comportamiento físico del circuito, adquiere una visión más profunda y completa del funcionamiento de un sistema robótico.

Ejercitando mis conocimientos

De manera individual y con la guía de tu profesor realiza lo siguiente:
Inicia un nuevo circuito en *Tinkercad* que resuelva lo para realizar un sistema simulado de semáforo simple con LED.

1. Coloca una placa **Arduino UNO R3**, además de un protoboard en el área de trabajo de *Tinkercad*.

2. Coloca tres LED al *protoboard*, verificando la polaridad de cada LED. Recuerda que ánodo es positivo y cátodo es negativo. Además de cambiar el color de los LED (amarillo, verde y rojo).

3. Coloca una resistencia de 220Ω por cada LED

4. Asigna un pin digital diferente por cada LED, por ejemplo:

- **a.** LED Verde al pin 6
- **b.** LED Amarillo al pin 7
- **c.** LED Rojo al pin 8

5. Realiza la conexión de los cátodos de los LED al riel del GND del protoboard y después realiza la conexión de tierra GND de Arduino.

6. Clic en el editor de código en *Tinkercad* y en la estructura de void *setup()* definiremos que estamos utilizando el pin 6,7 y 8. Para ello escribiremos los códigos para cada uno *pinMode(6, OUTPUT)*, después *pinMode(7, OUTPUT)*. y por último *pinMode(8, OUTPUT)*.

7. En la estructura *void loop()* agregaremos primeramente las siguientes líneas de código *digitalWrite(6,HIGH)* y *digitalWrite(8,LOW)*, para posteriormente la línea *delay(10000)*.

8. Después las siguientes dos líneas de códifo serían *digitalWrite(7,HIGH)* y *digitalWrite(6,LOW)* y posteriormente la línea *delay(2000)*.

9. Agrega la siguientes últimas líneas de código en la misma función *void loop()*: *digitalWrite(8,HIGH)* y *digitalWrite(7,LOW)*, posteriormente agrega la línea *delay(5000)*.

10. Una vez concluida la construcción del circuito, ejecuta la simulación y verifica que su funcionamiento sea el adecuado.

11. Exporta el circuito en formato de imagen y en un documento inserta la imagen y el código creado en *Tinkercad*.

12. Coloca el link del proyecto de *Tinkercad* en una sección del documento.

13. Guarda el documento colocando en el nombre del archivo tus iniciales seguidas de **_PC_P5_E03**.

14. Hazle llegar a tu profesor el documento por el medio que acuerden para recibir retroalimentación.

5.4 Sensores y actuadores

En la robótica educativa, los sensores y actuadores constituyen elementos esenciales que permiten al robot detectar su entorno y responder mediante acciones físicas. Estos componentes funcionan como el vínculo entre el mundo real y el sistema de control (por ejemplo, una placa Arduino), lo que hace posible desarrollar proyectos interactivos, experimentales y autónomos.

Un sensor es un dispositivo capaz de captar magnitudes físicas, como luz, temperatura, distancia, movimiento o presión, y transformarlas en señales eléctricas que pueden ser interpretadas por un **microcontrolador**. Debido a esta conversión, el robot obtiene información del entorno y puede tomar decisiones basadas en ella.

En entornos de simulación como *Tinkercad*, es posible conectar un sensor ultrasónico a un Arduino para medir distancias y, a partir de esa lectura, activar un motor, encender un LED o ejecutar cualquier acción programada. Esta interacción permite que se comprenda de manera guiada y visual cómo los datos proporcionados por los sensores determinan el comportamiento del robot.

Sensores

Los sensores, también conocidos como transductores, son dispositivos capaces de captar una magnitud física del entorno y convertirla en una señal eléctrica que pueda ser interpretada por un sistema de control.

Es decir, cada sensor está diseñado para detectar un tipo de estímulo específico, como luz, temperatura, movimiento, humedad, presión u otros fenómenos ambientales, proporcionando así los datos necesarios para que el sistema responda de manera adecuada.

Tipo y uso de sensores en Arduino

- **Sensor de temperatura:** permite medir la temperatura del entorno y generar una señal eléctrica proporcional a dicha variación. Uno de los modelos más utilizados es el **TMP36**, debido a su precisión y facilidad de uso en proyectos educativos y de automatización. Sus aplicaciones más comunes incluyen el monitoreo de temperatura en interiores, termostatos inteligentes, alarmas contra incendios, sistemas de refrigeración en vehículos y procesos de control en agricultura automatizada. Este sensor puede registrar valores dentro de un rango aproximado de –40 °C a 125 °C, lo que lo hace adecuado para diversos entornos.

- **Sensor de luz:** el modelo más utilizado es la **fotoresistencia**, también conocida como **LDR** (Light Dependent Resistor). Este componente está fabricado con materiales semiconductores cuya conductividad varía según la cantidad de luz que reciben: a mayor iluminación, menor resistencia, y a menor iluminación, mayor resistencia.

Sus aplicaciones abarcan desde la detección de presencia o ausencia de luz en una habitación, la regulación automática de cámaras fotográficas, la recepción de señales infrarrojas en controles remotos, hasta sistemas de iluminación automática

Conceptos clave

Microcontrolador. Pequeño circuito integrado que funciona como el **cerebro** de un dispositivo electrónico. Integra procesador, memoria y puertos de entrada y salida en un solo chip, lo que le permite recibir información mediante sensores, procesarla y activar actuadores según el programa que tenga cargado.

TMP36. Sensor de temperatura analógico que convierte los cambios de temperatura en una señal eléctrica proporcional, permitiendo medir valores del entorno de forma precisa y sencilla.

Fotoresistencia. Componente cuya resistencia varía según la cantidad de luz que recibe: disminuye con mayor iluminación y aumenta cuando hay poca luz. Este comportamiento la hace ideal para sistemas que responden automáticamente a la intensidad luminosa.

Conceptos clave

Transductor. Elemento que transforma una magnitud física (como calor o presión) en otra forma de energía, normalmente una señal eléctrica. Muchos sensores funcionan gracias a un transductor.

Rango de medición. Intervalo mínimo y máximo en el que un sensor puede realizar lecturas útiles sin perder precisión.

Sensor analógico. Emite señales continuas cuyo valor varía gradualmente, por ejemplo un LDR, un potenciómetro o el TMP36.

Sensor digital. Produce señales discretas, generalmente de tipo encendido/apagado o valores específicos. Ejemplo: sensor PIR de movimiento o módulos ultrasónicos.

Precisión. Grado de exactitud con el que un sensor mide una magnitud respecto a su valor real.

Relaciónalo con...

En la Progresión 5 de Cultura Digital 3, trabajaste con *Scratch* usando bloques de la categoría Sensores, como **tocando color**, **distancia a...** o **ruido**, para que tu personaje reaccionara al entorno digital.

En robótica ocurre algo muy parecido, solo que, en lugar de un personaje en la pantalla, es un robot real el que responde al mundo físico.
Así como *Scratch* detecta cambios en el juego, un sensor ultrasónico mide distancias, una fotoresistencia detecta luz y un **DHT11** identifica humedad o temperatura.

que encienden o apagan las luces según la intensidad luminosa del entorno. Existen diversos modelos de **fotoresistencias**, cada uno con un nivel de resistencia específico que responde de manera diferente a la cantidad de luz incidente.

- **Sensor de proximidad:** permite detectar la presencia o distancia de objetos cercanos, generando una señal en función de la medición realizada. El modelo más común es el **HC-SR04**, un sensor ultrasónico capaz de medir distancias entre 2 y 450 cm con una precisión aproximada de 3 mm. Está conformado por dos transductores ultrasónicos: un emisor y un receptor. Entre sus aplicaciones más frecuentes se encuentran la medición de distancias, la detección de objetos, la verificación del nivel de líquidos, el mapeo de espacios y la evitación de obstáculos en robots móviles.

- **Sensor de humedad:** detecta la humedad relativa del aire o de ciertos materiales y convertirla en una señal eléctrica interpretable por un microcontrolador. El modelo más utilizado es el **DHT11**, que incorpora un sensor de temperatura y un sensor de humedad. Sus rangos de medición van de 0 °C a 50 °C en temperatura y de 20 % a 90 % en humedad, aunque presenta algunas limitaciones en precisión. Es común emplearlo en estaciones meteorológicas caseras, sistemas de control ambiental, automatización del hogar y proyectos de IoT. Para aplicaciones que requieren mayor precisión, existen modelos más avanzados como el **DHT21** y el **DHT22**.

- **Sensor de sonido:** permite detectar variaciones en la presión del aire producidas por ondas sonoras y las convierte en señales eléctricas que pueden ser procesadas por Arduino. Los modelos más comunes son el **KY-037** y el **KY-038**. Entre sus aplicaciones destacan el control de luces mediante sonido, la creación de instrumentos musicales electrónicos, sistemas de seguridad y alarmas, mediciones de ruido ambiental y proyectos interactivos.

- **Sensor de gas:** detecta la presencia y concentración de gases inflamables y de humo. El modelo **MQ-2** es uno de los más utilizados y funciona gracias a un material interno sensible a los gases, cuya conductividad eléctrica cambia al entrar en contacto con ellos, esta variación puede ser medida por el sensor y procesada por Arduino para activar alarmas o sistemas de ventilación.

- **Sensor de vibración:** actúa como un interruptor sensible a impactos o movimientos bruscos, se activa al detectar vibraciones y se desactiva al mantener reposo. El modelo **SW-18015P** es uno de los más empleados y se utiliza en dispositivos de juego, sistemas de alarma, juguetes electrónicos, electrodomésticos y aplicaciones automotrices.

- **Sensor infrarrojo (IR):** mide la radiación infrarroja emitida por los objetos para detectar movimiento o presencia, se utiliza en sistemas de seguridad, iluminación automática y domótica para identificar personas o animales. Se clasifica como un sensor "pasivo" porque no emite energía propia; únicamente recibe la radiación infrarroja del entorno y la interpreta para generar una señal.

Conocer el funcionamiento y la correcta aplicación de los sensores es esencial para que un robot pueda interpretar su entorno y responder adecuadamente, los sensores proporcionan los datos que el microcontrolador necesita para ejecutar acciones precisas, por lo que dominar su uso permite diseñar proyectos de robótica más seguros, confiables y funcionales.

Tipo y uso de sensores en Arduino

Sensor	Funciones comunes	Imagen
Ultrasonico (HC-SR04)	Detección de objetos, evitar obstáculos, medir niveles.	
Fotoresistencia (LDR)	Control de iluminación, alarmas luminosas, medición de brillo.	
Temperatura (TMP36)	Monitoreo térmico, termostatos, control ambiental.	
Humedad y temperatura (DHT11)	Estaciones meteorológicas, sistemas automatizados, IoT.	
Sensor de sonido	Alarmas, proyectos interactivos, detección de ruido.	
Sensor infrarrojo (IR)	Seguridad, iluminación automática, domótica.	
Sensor de gas (MQ-2)	Alarmas de gas, sistemas de ventilación, monitoreo ambiental.	
Sensor de vibración (SW-18015P)	Alarmas, juguetes electrónicos, dispositivos interactivos.	

Por lo general, al desarrollar proyectos que incorporan sensores, no se requiere el uso de bibliotecas específicas para su funcionamiento. Sin embargo, existen ciertos dispositivos, como los sensores de temperatura y humedad DHT11 o DHT12, que sí demandan la utilización de la biblioteca DHT.h. De manera similar, algunos sensores más especializados, como los acelerómetros y giroscopios, requieren las bibliotecas Wire.h y MPU6050.h, respectivamente. Estas bibliotecas deben incluirse en las primeras líneas del código del proyecto, de forma análoga a como se procede en el lenguaje C++.

Relaciónalo con...

En *Tinkercad* también es posible trabajar con librerías adicionales para sensores avanzados, igual que en Arduino físico. Para hacerlo, basta con escribir manualmente en el código las líneas que incluyen cada biblioteca, por ejemplo: **#include <DHT.h>** o **#include <Wire.h>.**

Para saber más...

Escanea el código QR y observa el video Sensores en *Arduino* (ultrasónico).

Para saber más...

Escanea el código QR y observa el video Sensores en *Arduino* (proximidad, luz, humedad).

Ejercitando mis conocimientos

De manera individual y siguiendo las indicaciones de tu profesor, realiza lo siguiente: Crea un nuevo circuito en *Tinkercad* para simular un sistema que encienda un LED automáticamente cuando el entorno se oscurezca.

1. Coloca una placa Arduino UNO R3 y un protoboard en el área de trabajo de *Tinkercad*.

2. Inserta un LED en el *protoboard* y verifica su polaridad: recuerda que el ánodo es positivo y el cátodo es negativo. Cambia el color del LED a tu preferencia.

3. Añade una resistencia de 220 Ω al LED para limitar la corriente y evitar daños al componente.

4. Coloca una fotoresistencia (LDR) en el *protoboard*. Este sensor permitirá detectar los niveles de luz del entorno.

5. Conecta la LDR formando un divisor de voltaje:

- **a.** Conecta uno de sus terminales a 5V.
- **b.** Conecta el otro terminal a una resistencia de 10 kΩ hacia GND.
- **c.** Desde el punto donde se unen la LDR y la resistencia, lleva un cable hacia el pin A0 de Arduino (lectura analógica).

6. Conecta el LED a un pin digital de Arduino, por ejemplo:

- **a.** LED → pin 7
- **b.** Cátodo del LED → riel GND del protoboard
- **c.** Conecta también el GND de Arduino al riel de tierra del protoboard.

7. Abre el editor de código en *Tinkercad*. En la función void *setup()*, configura el pin 7 como salida y declara el uso del pin analógico A0:

- **a.** *pinMode(7, OUTPUT)*;

8. En la función void *loop()*, escribe las líneas de código necesarias para:

- **a.** Leer el valor del sensor con *analogRead(A0)*;
- **b.** Encender el LED cuando el valor sea bajo (oscuridad)
- **c.** Apagarlo cuando el valor sea alto (luz)

9. Ejecuta la simulación y ajusta la iluminación del entorno con las herramientas de *Tinkercad* para verificar que el LED se enciende cuando hay oscuridad y se apaga cuando hay luz.

10. Una vez validado el funcionamiento, exporta en formato de imagen y crea un documento donde insertes la imagen del circuito y el código utilizado.

11. Coloca el link del proyecto de *Tinkercad* en una sección del documento.

12. Guarda el documento nombrándolo con tus iniciales seguidas de **_PC_P5_E04**.

13. Entrega el documento a tu profesor por el medio que acuerden para recibir retroalimentación.

Actuadores

Un actuador es un dispositivo que recibe una señal de control generalmente eléctrica y la transforma en una acción física, ya sea movimiento, sonido, luz o vibración. Los actuadores pueden clasificarse según la fuente de energía o el principio físico que emplean:

- **Eléctricos:** incluyen motores de corriente directa (DC), servomotores, relés y dispositivos vibradores.
- **Neumáticos:** funcionan mediante aire comprimido para generar movimiento.
- **Hidráulicos:** utilizan líquidos a presión para producir fuerza o desplazamiento.

Los actuadores también pueden clasificarse según la función que desempeñan dentro de un sistema robótico:

- **Motores DC:** convierten la energía eléctrica en un movimiento rotacional continuo, su funcionamiento se basa en la interacción entre un campo magnético y una corriente eléctrica, lo que genera el giro constante del eje.

Son utilizados principalmente para desplazar ruedas, accionar mecanismos sencillos o producir movimientos repetitivos sin necesidad de un control preciso de posición, comúnmente usados en robots móviles y sistemas que requieren desplazamiento sostenido.

- **Servomotores:** están diseñados para controlar con gran precisión la posición angular de un eje a diferencia de los motores DC, incorporan un sistema de retroalimentación que permite conocer y corregir su posición en todo momento. Esto los hace esenciales para brazos robóticos, mecanismos articulados y sistemas de dirección, donde se requiere estabilidad, suavidad y exactitud en el movimiento. Su capacidad para mantener una posición fija ante pequeñas perturbaciones es una de sus principales ventajas.

- **Relés:** funcionan como interruptores controlados eléctricamente que permiten activar o desactivar circuitos de mayor potencia mediante señales de bajo voltaje provenientes de un microcontrolador, como Arduino. Operan a través de un electroimán que, al recibir corriente, acciona un contacto interno y modifica el estado del circuito externo, gracias a este mecanismo, los relés proporcionan aislamiento eléctrico y protección para los componentes sensibles, y son ideales para controlar lámparas, motores grandes, electrodomésticos u otros dispositivos que no pueden conectarse directamente al sistema de control.

- **Bombas, válvulas y actuadores lineales:** estos dispositivos transforman la energía eléctrica en movimientos específicos, como empujar, tirar, abrir o cerrar mecanismos. Las bombas permiten desplazar líquidos o gases y se utilizan en sistemas de riego o proyectos hidráulicos; las válvulas regulan el flujo de fluidos en tuberías, mientras que los actuadores lineales convierten la energía eléctrica en un desplazamiento recto, útil en sistemas que requieren levantar o posicionar objetos con precisión. Estos componentes son comunes en proyectos avanzados o aplicaciones industriales.

- **Indicadores luminosos o sonoros:** los indicadores como LED y zumbadores cumplen la función de comunicar información al usuario mediante luz o sonido. Aunque no generan movimiento físico, resultan esenciales para señalizar estados, emitir alertas o indicar procesos en curso. Los LED pueden encenderse, apagarse o variar su intensidad, mientras que los zumbadores producen señales acústicas útiles para advertencias y notificaciones. Su sencillez y bajo consumo energético los hace ampliamente utilizados en proyectos educativos e interactivos.

Para saber más...

Escanea el código QR y observa el video Actuadores en *Arduino*.

Para saber más...

Escanea el código QR y observa el video Servomotores en *Arduino*.

En el ámbito de la robótica educativa, un ejemplo representativo es el uso de un servomotor **SG90** controlado mediante Arduino para mover un brazo robótico, o la activación de un LED o un zumbador cuando un sensor de distancia detecta un obstáculo.

Tipo y uso de actuadores en Arduino

Sensor	Funciones comunes	Imagen
DC Motor	Produce movimiento rotacional continuo. Se usa en robot móviles, ventiladores, mecanismos simples y ruedas motrices.	
Servomotor (SG90)	Controla la posición angular con alta precisión. Ideal para brazos robóticos, mecanismos articulados, dirección de robots y sistemas que requieren movimientos exactos.	
Realy	Funciona como interruptor de alta potencia controlado por una señal de bajo voltaje. Se utiliza para encender lámparas, activar motores grandes, controlar electrodomésticos y aislar circuitos.	
Zumbador piezoeléctrico	Produce sonidos o alertas acústicas. Se usa en alarmas, notificaciones, sistemas interactivos y señales audibles.	
LED / Indicador luminoso	Emite luz para señalización de estados, advertencias, indicadores visuales o retroalimentación de procesos.	

En la mayoría de los proyectos con Arduino, los actuadores básicos no requieren bibliotecas adicionales para su funcionamiento componentes como LEDs, zumbadores o relés pueden controlarse directamente mediante instrucciones estándar, como *digitalWrite()* o *analogWrite()*, sin necesidad de configuraciones especiales.

Ejercitando mis conocimientos

De manera individual y siguiendo las indicaciones de tu profesor, realiza lo siguiente:

1. Creen un nuevo circuito en *Tinkercad*.

2. Coloca una placa Arduino UNO R3 y un *protoboard* en el área de trabajo de *Tinkercad*.

3. Escanea el código QR del lado izquierdo para descargar el archivo con las indicaciones detalladas.

4. Exporta una imagen del circuito y elabora un documento donde insertes la imagen y el código utilizado.

5. Coloca el link del proyecto de *Tinkercad* en una sección del documento.

6. Guarda el documento con el nombre correspondiente a tus iniciales seguido de **_PC_P5_E05**.

7. Envía tu archivo al profesor mediante el medio que hayan acordado para recibir retroalimentación.

Recurso digital

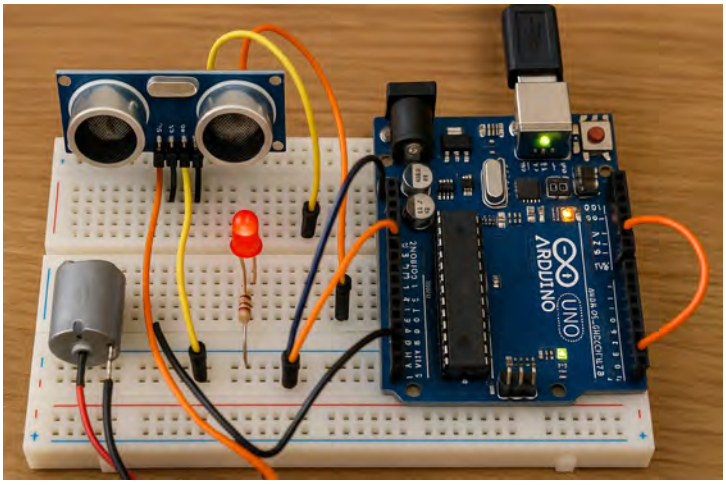
Escanea el código QR con las instrucciones detalladas para un sistema de ventilador automatizado con sensor de temperatura.

Integración de sensores y actuadores

La integración de sensores y actuadores constituye uno de los aspectos esenciales en el desarrollo de sistemas robóticos, ya que permite crear dispositivos capaces de percibir su entorno y responder mediante acciones físicas. Un sensor proporciona información del mundo real, mientras que un actuador transforma esa información en una acción concreta, como mover un motor, encender un LED o activar un mecanismo.

Esta interacción convierte un circuito sencillo en un sistema autónomo capaz de ejecutar tareas de manera automática. En un proyecto robótico, la placa Arduino actúa como el elemento central que recibe las lecturas del sensor, procesa esos datos mediante el programa cargado y envía órdenes precisas al actuador correspondiente. De esta forma, la lógica del programa se convierte en el puente entre el análisis del entorno y la respuesta física del sistema.

En *Tinkercad* es simular cómo cambian las lecturas de un sensor en tiempo real y verificar de inmediato la reacción del actuador asociado, ya sea el encendido de un LED, la activación de un motor o la emisión de un sonido. Este proceso facilita la comprensión de la relación entre *hardware* y *software*. La simulación de *Tinkercad* también permite experimentar con sistemas como iluminación automática, ventilación inteligente, alarmas sonoras, medidores ambientales o mecanismos de movimiento básico, identificando cómo las variaciones en las condiciones del entorno influyen en el comportamiento del sistema.



Proyecto de robótica con integración de sensores y actuadores.

La integración de sensores y actuadores no solo fortalece las habilidades técnicas del estudiante, sino que también fomenta el razonamiento lógico, la creatividad y la resolución de problemas. Al comprender cómo interactúan estos componentes dentro de un circuito y cómo el código determina su comportamiento, el desarrollador desarrolla una visión más completa del funcionamiento de la robótica y se prepara para enfrentar proyectos más avanzados tanto en simulación como en hardware físico.

Para saber más...

Escanea el código QR y observa el video Sistema integral básico.

Recurso digital

Escanea el código QR con las instrucciones detalladas para armar el sistema de alarma inteligente.

Demostrando mi aprendizaje

Para demostrar tu aprendizaje conceptual referente a los temas abordados en esta progresión, realiza la actividad interactiva, ingresa a ella escaneando el código QR.

Concretando mis conocimientos

Reúnete con tres compañeros más y de manera colaborativa realicen un proyecto en *Tinkercad* integrando sensores y actuadores para crear un sistema de alarma inteligente.

1. Creen un nuevo circuito en *Tinkercad*.

2. Coloca una placa Arduino UNO R3 y un protoboard en el área de trabajo de *Tinkercad*.

3. Escanea el código QR del lado derecho para descargar el archivo con las indicaciones detalladas.

4. Ejecuta la simulación y ajusta la temperatura desde el panel del sensor para verificar que el ventilador se active cuando la temperatura aumenta y se desactive cuando baja.

5. Exporta una imagen del circuito y elabora un documento donde insertes la imagen y el código utilizado.

6. Coloca el link del proyecto de *Tinkercad* en una sección del documento.

7. Guarda el documento con el nombre correspondiente número de equipo seguido por _**PC_P5_CMC**.

8. Envíen su archivo al profesor mediante el medio que hayan acordado para recibir evaluación.

Instrumento de evaluación

Revisa la siguiente lista de cotejo para que conozcas los criterios con los que tu profesor evaluará tu programa en *Tinkercad*.

Indicador	Si	No	Puntos
El circuito está correctamente armado en *Tinkercad* (*Arduino*, *protoboard*, sensor y actuadores bien colocados).).			2
Las conexiones eléctricas son correctas (polaridad, resistencias, GND/5V, entrada del sensor)..			2
El código configura correctamente los pines en setup() y realiza la lectura del sensor.			2
La alarma responde adecuadamente a la condición definida (se activa y desactiva según el sensor).			2
El documento entregado contiene la imagen del circuito, el código completo y una breve explicación del funcionamiento.			2

Valorando mi aprendizaje

La evaluación es un proceso continuo de formación, útil para recabar evidencias sobre el logro de los aprendizajes, con oportunidad de retroalimentación y mejora de los resultados.

En este apartado se presentan algunas actividades e instrumentos, que te guían en la valoración de los aprendizajes que adquiriste progresivamente en las secuencias didácticas anteriores. Responde honestamente a cada una de ellas.

Reflexionando lo que aprendí

Contesta las siguientes preguntas y reflexiona sobre tu desempeño en esta última progresión..

- Describe una situación de tu vida académica donde podrías aplicar lo aprendido en robótica educativa para organizarte, resolver un problema o automatizar una tarea.

- Piensa en un proyecto de robótica que realizaste. ¿Qué fue lo más retador al integrar sensores y actuadores y cómo resolviste ese reto?

- ¿Cómo te ayudó trabajar con Arduino a entender mejor cómo funcionan los dispositivos electrónicos en la vida real? Explica con un ejemplo.

- Después de estudiar robótica educativa, ¿qué consideras que necesitas mejorar o seguir practicando y por qué crees que esto será importante para tu futuro académico o profesional?

Actividad alternativa

Resuelve para reforzar tu aprendizaje e incrementar tu evaluación.

Indicaciones:

1. Observa tu entorno durante 1 o 2 días. Pon atención en cualquier elemento donde intervengan robots, automatizaciones, sensores, actuadores o dispositivos inteligentes.

2. Investiga y recopila un dato curioso. Puedes investigar mediante observación, Internet, entrevistas o aparatos reales.

3. Crea un video donde lo expliques a profundidad. Incluye los siguientes datos: cómo funciona, sensores involucrados, problema que soluciona y por qué te llamó la atención.

4. Aplica un diseño creativo y explica con claridad y orden.

5. Entrega el video a tu profesor para que evalúe.

Autoevaluación

La autoevaluación es un mecanismo de autocontrol que te ayuda a regular tu aprendizaje. Marca con una √ la columna que corresponda a tu nivel de dominio en los aspectos de aprendizaje en cada meta.

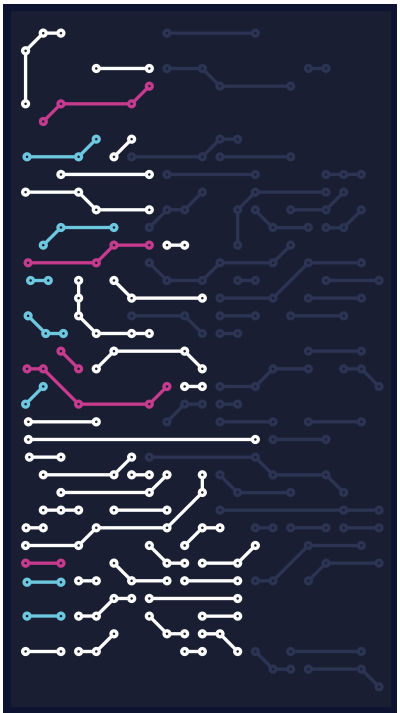
Metas	Criterios	Nivel de dominio		
		Sí lo logro	En proceso	Aún no lo logro
Identifica la importancia de la robótica educativa como herramienta para comprender la interacción entre hardware y software.	Explico qué es la robótica educativa y su utilidad en el aprendizaje tecnológico.			
	Reconozco la relación entre hardware y software en proyectos de robótica.			
	Describo la forma en que Tinkercad y Arduino representan la interacción hardware–software.			
Configura sistemas básicos de automatización en un entorno gráfico controlando las salidas.	Configuro circuitos básicos en Tinkercad con componentes electrónicos.			
	Conecto correctamente salidas (actuadores) como LED, servo o buzzer en Arduino.			
	Utilizo el simulador para verificar que las salidas reaccionen según el diseño.			
	Comprendo el flujo de energía y señal entre la placa Arduino y los actuadores.			
	Ajusto parámetros (ángulo de servo, intensidad, frecuencia) cuando el circuito lo requiere.			
Codifica programas con estructuras de control, funciones básicas, bucles, sensores y actuadores en simulaciones.	Escribo código Arduino que controla salidas mediante *digitalWrite*, *analogWrite* o funciones específicas.			
	Implemento estructuras de control para responder a condiciones del entorno.			
	Programo sensores básicos (LDR, ultrasonido, botón, potenciómetro) e interpreta sus lecturas.			
	Integro sensores y actuadores en una simulación que responda a entradas reales.			
	Depuro y corrijo errores en el código hasta lograr una simulación funcional.			
Lo mejor que aprendí fue:				
Lo que necesito reforzar es:				

Calificación que doy a mi desempeño:	Excelente	Satisfactorio	En desarrollo	Inicial

Coevaluación

Evalúa el desempeño general de tu equipo de trabajo durante el desarrollo de las actividades de aprendizaje colaborativas. Coloca el valor correspondiente en la columna Evaluación y suma para conocer el resultado del trabajo por equipo.

Buen trabajo (3)	Algo nos faltó (2)	Debemos mejorar (1)	Evaluación
Organizamos el trabajo estipulando tareas, prioridades y plazos.	Se organizó el trabajo, pero no se estipularon tareas, prioridades o el plazo de entrega final.	No hubo organización para realizar nuestros trabajos.	
Cumplimos cada uno con las tareas asignadas en el plazo estipulado.	Casi todos los miembros del equipo cumplimos con las tareas asignadas y el plazo estipulado; teniendo que resolver lo que a otros les fue encomendado.	Un solo miembro del equipo realizó todos los productos.	
Todos participamos activamente en la elaboración de los productos.	Casi todos los miembros del equipo participamos activamente en la elaboración de los productos.	No hubo participación de los miembros del equipo en la elaboración de los productos.	
La calidad de los productos que elaboramos fue la adecuada para su entrega.	La calidad de los productos que elaboramos fue en su mayoría la adecuada para su entrega.	No se cumplió con la calidad adecuada de los productos para su entrega.	
		Total	___ de 12



Cairó, O. (2005). Metodología de la programación. México: Alfaomega.

Cairó, O. (2007). Metodología de la programación para Bachillerato. Alfaomega. México.

Celi, P. (2023). Fundamentos de programación basados en PSeInt. Quito: Doxa Edition.

De Anda, C., Santiago, R., & Romero, E. (2024). Tecnologías de la información 3: Laboratorio de cómputo III (2.ª ed.). Dirección General de Escuelas Preparatorias-UAS. Ediciones GYROS, S. A. de C. V. México.

De Anda, C., Santiago, R., & Romero, E. (2020). Introducción a la programación: Laboratorio de cómputo IV (1.ª ed.). Dirección General de Escuelas Preparatorias-UAS. Ediciones GYROS, S. A. de C. V. México.

Deitel P. & Deitel H. (2014). C++ Cómo programar. (9.ª ed.). Pearson Educación, México.

Díaz-Brito, L., & Rodríguez-Guzmán, A. (2018). Pseudocódigo y programación estructurada para secundaria y bachillerato (2.ª ed.). Editorial Limusa. México.

García-Molina, J., & Pérez-Campos, F. (2020). Introducción al pensamiento computacional: Algoritmos, pseudocódigo y resolución de problemas. Ediciones Alfaomega. México.

Joyanes, L. (2008) Fundamentos de la programación. España: McGraw-Hill

SEP (2023a). Progresiones de aprendizaje del recurso sociocognitivo Cultura digital. SEMS. Secretaría de Educación Pública, Subsecretaría de Educación Media Superior. Segunda edición. Consultado el 18 de diciembre del año 2023 en: https://educacionmediasuperior.sep.gob.mx/work/models/sems/Resource/13634/1/images/Progresiones%20de%20aprendizaje%20-%20Cultura%20Digital(1).pdf

Shamieh, C. (2015). Electronics for Dummies. United States of America: John Wiley & Sons, Inc.

Stroustrup, B. (2014). Programming: Principles and Practice Using C++. Crawfordsville, Florida: Addison-Wesley.

UAS (2022). Modelo educativo Universidad Autónoma de Sinaloa.

UAS (2024). Currículo del Bachillerato DGEP-UAS. Culiacán Rosales, Sinaloa.

Fuentes digitales

https://es.slideshare.net/slideshow/tinkercad-practicas-y-soluciones/250056857

https://www.tinkercad.com/blog/official-guide-to-tinkercad-circuits

PENSAMIENTO COMPUTACIONAL

Pensamiento Computacional para el Bachillerato es un libro diseñado con los lineamientos del programa de estudio del mismo nombre del Plan Bachillerato UAS 2024 de la Universidad Autónoma de Sinaloa. El texto se orienta bajo los enfoques humanista y constructivista del Modelo Educativo UAS 2022, siguiendo también las directrices del Marco Curricular Común de la Educación Media Superior de la Nueva Escuela Mexicana, que busca formar una sociedad preparada para los desafíos del presente y futuro.

El libro está estructurado en cinco progresiones de aprendizaje, las cuales buscan que los estudiantes adquieran conocimientos y habilidades para generar soluciones lógicas y eficientes. Las progresiones incluyen:

Pensamiento computacional y algoritmos básicos, donde los estudiantes aprenderán a descomponer problemas complejos; **Algoritmia en IDE**, que los familiarizará con entornos de desarrollo; y **Programación estructurada en C++: Estructuras de control y Estructuras de datos**, para que dominen la sintaxis y organización de datos en la programación. Finalmente, la progresión de **Robótica en simuladores virtuales** les permitirá aplicar estos conceptos en proyectos prácticos, fomentando así la creatividad y la capacidad de innovación.

Pensamiento Computacional fortalece la formación integral de los estudiantes del Bachillerato de la UAS, ayudándoles a reconocer la importancia del desarrollo de habilidades tecnológicas para buscar, comunicar, investigar e interactuar en entornos digitales. Mediante proyectos y actividades colaborativas e individuales, el libro promueve el desarrollo de un pensamiento crítico-reflexivo, permitiendo a los estudiantes diseñar y elaborar contenidos digitales y soluciones tecnológicas que les serán útiles en su vida cotidiana. El objetivo final es que los jóvenes se conviertan en individuos conscientes del uso ético y responsable de las Tecnologías de la Información, Comunicación, Conocimiento y Aprendizaje Digitales (TICCAD), capaces de generar nuevo conocimiento y promover la reflexión crítica en su entorno.

GYROS EDITORIAL

ISBN 978-970-96930-4-1

9 789709 693041