

UNIVERSIDAD NACIONAL DE EDUCACIÓN

Enrique Guzmán y Valle

Alma Máter del Magisterio Nacional

FACULTAD DE CIENCIAS

Escuela Profesional de Matemática e Informática



MONOGRAFÍA

INTRODUCCIÓN A LA PROGRAMACIÓN

Introducción, fundamentos de la Programación, herramientas de programación, metodología de Programación, aplicaciones.

Examen de Suficiencia Profesional Res. N°0505-2019-D-FAC

Presentada por:

Caceres Espinoza, Liliana

Para optar al Título Profesional de Licenciado en Educación

Especialidad: Informática

Lima, Perú

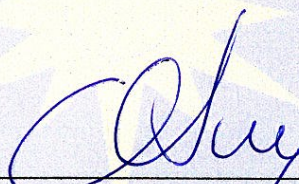
2019

MONOGRAFÍA

INTRODUCCIÓN A LA PROGRAMACIÓN

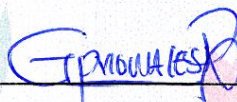
Introducción, fundamentos de la Programación, herramientas de programación, metodología de Programación, aplicaciones.

Designación de Jurado Resolución N°0505-2019-D-FAC



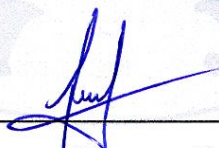
Dr. Caballero Cifuentes, Lolo José

Presidente



Dr. Morales Romero, Guillermo Pastor

Secretario



Dra. Vargas Tumaya, Jady Luz

Vocal

Línea de investigación: Tecnología y soportes educativos.

A mi familia, por el apoyo incondicional que me
brindaron durante el proceso de mi formación
profesional.

Índice de contenidos

Portada.....	i
Hoja de firmas de jurado.....	ii
Dedicatoria.....	iii
Índice de contenidos.....	iv
Lista de tablas.....	vi
Lista de figuras.....	vii
Introducción.....	viii
Capítulo I. Fundamentos de la programación.....	9
1.1 Conceptos principales.....	9
1.1.1 Programación.....	9
1.1.2 Algoritmo.....	10
1.1.3 Programa.....	10
1.1.4 Lenguaje de programación.....	10
1.2 Lenguajes de programación.....	10
1.2.1 Tipos de lenguaje.....	11
1.2.1.1 Lenguajes de máquina y ensamblaje.....	11
1.2.1.1.1 Máquina	11
1.2.1.1.2 Bajo nivel.....	11
1.2.1.1.3 Alto nivel.....	12
1.3 Traductores.....	12
1.3.1 Compiladores.....	12
1.3.2 Intérpretes.....	13
1.4 Fases de la programación.....	15

1.4.1 Definición de problema	15
1.4.2 Análisis de problema.....	16
1.4.3 Diseño y elaboración del algoritmo.....	16
1.4.4 Codificación.....	18
1.4.5 Prueba y depuración (puesta a punto o testing).....	18
1.4.6 Documentación.....	20
1.4.7 Implementación (producción)	21
1.4.8 Mantenimiento.....	21
Capítulo II. Herramientas de programación.....	22
2.1 Algoritmo.....	22
2.2 Pseudocódigo.....	24
2.3 Diagrama de flujo.....	25
Capítulo III. Metodología de programación.....	26
3.1 Diseño Top Down.....	26
3.2 Programación modular.....	26
3.3 Programación estructurada.....	28
3.4 Programación no estructurada.....	30
3.5 Programación orientada a objetos (POO).....	31
Aplicación didáctica.....	37
Síntesis.....	43
Apreciación crítica y sugerencias.....	44
Referencias.....	45

Lista de tablas

Tabla 1. Clasificación de lenguaje de programación.....	14
---------------------------------------------------------	----

Lista de figuras

Figura 1. Proceso traductor de programas.....	12
Figura 2. Compilación (fases de la compilación).....	13
Figura 3. Intérpretes.....	14
Figura 4. Diagrama de flujo.....	17
Figura 5. Pseudocódigo ejemplo de estructura selectiva simple.....	24
Figura 6. Símbolos del diagrama de flujo.....	25

Introducción

La monografía titulada *Introducción a la programación* desarrolla el marco teórico para comprender sobre programación como un proceso mediante el cual se codifican una serie de instrucciones, en un determinado lenguaje, para ser posteriormente decodificados y ejecutados por un sistema computacional, todo ello con el fin de resolver un problema y que debe ser entendida por todo estudiante de educación superior.

Esta monografía servirá de referencia y guía de estudio, porque presenta los fundamentos, herramientas y metodologías más sobresalientes de la programación.

El trabajo tiene una organización en 3 capítulos: el capítulo I, expone sobre los fundamentos de la programación; el capítulo II, explica sobre las herramientas de programación; el capítulo III, trata sobre metodología de programación. Finalmente, se presenta la aplicación didáctica mostrando la sesión de aprendizaje, guía de laboratorio y un instrumento que permita valorar el aprendizaje, la síntesis, apreciación crítica y sugerencias, y referencias.

Capítulo I

Fundamentos de la programación

1.1 Conceptos principales

La computadora permite procesar datos para generar información, este proceso es en forma automática, para lo cual realiza muchas operaciones. Según Trembay (2011):

Pero del mismo modo se puede diseñar procesos a medida, de ciertas situaciones que nos apremian, los cuales pueden involucrar diversas operaciones sean aritméticas o lógicas y procesos de decisión y/o repetitivas, o volúmenes muy grandes de datos para su proceso (p.18).

1.1.1 Programación.

Es un proceso mediante el cual se codifican una serie de instrucciones, en un determinado lenguaje, para ser posteriormente decodificados y ejecutados por un sistema computacional, todo ello con el fin de resolver un problema. Es decir, implementar desde un algoritmo hacia un lenguaje de programación y dar solución a un problema.

1.1.2 Algoritmo.

Conjunto de pasos finitos y secuenciales que permiten resolver un problema. El nombre de algoritmos deriva del matemático Persa Alkhowarizmi en el siglo XVII.

1.1.3 Programa.

Es una secuencia de instrucciones que una computadora puede interpretar y ejecutar.

1.1.4 Lenguaje de programación.

Los lenguajes de programación son una serie de comandos que escritos secuencialmente y respetando la sintaxis resuelven una situación problemática en cuanto a información.

1.2 Lenguajes de programación

Lenguaje de programación, se expresan mediante un conjunto de instrucciones detalladas para una computadora digital. Dichas instrucciones se pueden ejecutar directamente cuando están en la forma numérica específica del fabricante de la computadora conocida como lenguaje de máquina, después de un simple proceso de sustitución cuando se expresa en un lenguaje ensamblador correspondiente, o después de la traducción de algún lenguaje de "nivel superior". Aunque hay muchos lenguajes de computadora, relativamente pocos son ampliamente utilizados.

Los lenguajes de máquina y ensamblaje son de "bajo nivel", lo que requiere que un programador administre explícitamente todas las características idiosincráticas de almacenamiento y operación de datos de una computadora. Por el contrario, los lenguajes de alto nivel protegen al programador de preocuparse por tales consideraciones y proporcionan una notación que los programadores escriben y leen más fácilmente.

1.2.1 Tipos de lenguaje.

1.2.1.1 Lenguajes de máquina y ensamblaje.

Un lenguaje de máquina consiste en los códigos numéricos para las operaciones que una computadora en particular puede ejecutar directamente. Los códigos son cadenas de 0s y 1s, o dígitos binarios (bits), que con frecuencia se convierten de y a hexadecimales (base 16) para la visualización y modificación humana. Las instrucciones del lenguaje de máquina generalmente usan algunos bits para representar operaciones, como la suma, y algunos para representar operandos, o tal vez la ubicación de la siguiente instrucción. El lenguaje de máquina es difícil de leer y escribir, ya que no se parece a la notación matemática convencional o al lenguaje humano, y sus códigos varían de una computadora a otra.

1.2.1.1.1 Máquina.

El lenguaje ensamblador está un nivel por encima del lenguaje máquina. Utiliza códigos mnemotécnicos cortos para las instrucciones y permite al programador introducir nombres para bloques de memoria que contienen datos. Por lo tanto, se podría escribir "agregar pago, total" en lugar de "0110101100101000" para una instrucción que agrega dos números.

1.2.1.1.2 Bajo nivel.

Generalmente los utilizan los mismos fabricantes y su comprensión es muy complicada, son llamados también lenguajes ensambladores traduciendo estos a órdenes que son ejecutados por la maquina (lenguaje de máquina). Son realizados bajos esquemas nemónicos.

Ej.	Mov A, 1, C	→ Instrucción 1
	Add A, B	→ Instrucción 2

1.2.1.1.3 Alto nivel.

Los lenguajes algorítmicos están diseñados para expresar cálculos matemáticos o simbólicos. Pueden expresar operaciones algebraicas en notación similar a las matemáticas y permiten el uso de subprogramas que empaquetan operaciones de uso común para su reutilización. Fueron los primeros idiomas de alto nivel.

1.3 Traductores

Son programas que traducen los programas fuente escritos en lenguajes de alto nivel a lenguaje de máquina y son: compiladores e intérpretes.

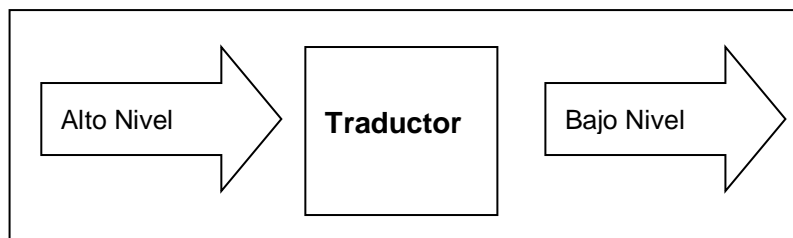


Figura 1. Proceso traductor de programas. Fuente: Autoría propia.

1.3.1 Compiladores.

Para Carrasco (2013) “un compilador es un programa que traduce los programas fuente escritos en lenguajes de alto nivel a lenguaje máquina” (p.26). Son llamados lenguaje fuentes a aquellos codificados en lenguajes de alto nivel y los ya traducidos se les llama lenguajes objetos, el compilador tiene la función de verificar cada instrucción y si hay problemas envía mensajes indicando las fallas, este proceso es repetitivo hasta que no tenga ningún error de sintaxis.

Lenguajes compiladores típicos son: Visual Basic, JAVA, COBOL, etc. Para Carrasco (2013):

El programa objeto obtenido de la compilación no ha sido traducido normalmente a código máquina sino a ensamblador. Para conseguir el programa máquina real se debe utilizar un programa llamado montador o enlazador (linker). El proceso de montaje conduce a un programa en lenguaje máquina directamente ejecutable (p.55).

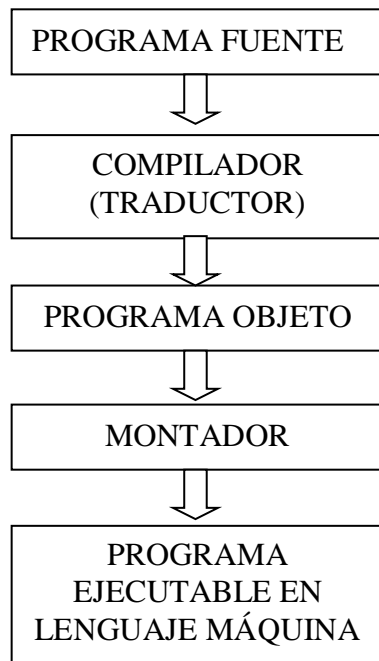


Figura 2. Compilación (fases de la compilación). Fuente: Autoría propia.

1.3.2 Intérpretes.

Carrasco (2013) mencionó que:

Un intérprete es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta (dicho programa por medio de la computadora desarrolla una tarea específica). A diferencia de los compiladores revisa y ejecuta instrucción por instrucción, de tal modo que puede presentar anomalías antes de finalizar la tarea (p.62).

Cuando un lenguaje de programación de alto nivel ha pasado por la etapa de traducción mediante un interpretador se le denomina lenguaje interpretado, siendo el BASIC el lenguaje representante de este modelo. La codificación de estos lenguajes, generalmente se realiza por sus propios editores. Tienen como principal característica que son traducidos y verificados línea a línea lo que permite corregir los errores inmediatamente hasta corregirlos totalmente, generando igualmente una versión compilada.

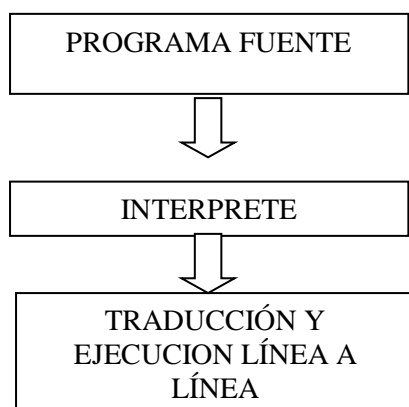


Figura 3. Intérpretes. Fuente: Autoría propia.

Tabla 1
Clasificación de lenguaje de programación.

Criterio de clasificación	Tipo LP	Ejemplo
Según su nivel de acción a la máquina.	L de alto nivel	SQL, Quel, FOCUS, Prolog
	L. Ensamblador	C, Pascal, Fortran, Ada, C++
	L. de Máquina	
	L. Generalizado	C, CC++, Ada, Modula-2
Según su dominio de aplicación	L. Científico	Fortran, APL, Pascal
	L. Comercial	Cobol, RPG
	L. de Datos	PS, Algol, SQL, Daplex, FDL
	L. Estadísticos	SAS, SPSS
	L. de Inteligencia Artificial	Prol, Lisp, CLOS
	L. de Simulación	Simula, GPSS, SIMSCRIPT
Según su modelo computacional paradigma o enfoque	L. Imperativo	C, Pascal, Modula-2
	L. Orientado a objetos	Smalltalk, Eiffel, C++, Java
	L. Funcional	Lisp, Scheme, ML
	L. de Programación Lógica	Prolog
	L. Concurrente	ADA, CSP, Pascal Concurrente
Según su desarrollo histórico	L. de 1a. Generación	Lenguajes de máquina
	L. de 2a. Generación	Ensambladores
	L. de 3a. Generación	Fortran, Algol, Pascal, C
	L. de 4a. Generación	FOCUS, SQL, Quel,
	L. de 5a. Generación	Prolog.

Nota: Principales lenguajes de programación. Fuente: Joyanes, 2006.

1.4 Fases de la programación

Antes de pasar a discutir los cinco pasos para crear un programa, es importante determinar qué es exactamente un programa. Un programa es una lista de instrucciones que contienen datos que debe seguir una computadora. Los diferentes programas se escriben con diferentes idiomas. Un programa de edición se realiza con un "lenguaje" de programación diferente al que usa gráficos.

La generación de un programa o un Software necesita una metodología como modelo para lograr realizar el algoritmo y resolver el problema. Esta metodología llamada Ciclo de desarrollo del Software, consta de una serie de pasos lógicos secuenciales denominados fases, son:

1. Definición del problema
2. Análisis del problema
3. Diseño de la solución
4. Codificación
5. Prueba y depuración (puesta a punto o testing)
6. Documentación
7. Implementación (producción)
8. Mantenimiento

1.4.1 Definición del problema.

Conocer el problema es la primera consideración (es una nómina o un programa de edición). Saber quién será el usuario final, también es importante. La determinación de las entradas y salidas es la siguiente. Cómo funcionará el programa y qué datos se necesitan para que esto suceda. Después de que esto se haya decidido, la factibilidad será la siguiente consideración. Cuantos programadores considerarán que si el proyecto tiene un esquema

realista. Finalmente, si el proyecto está listo entonces uno debe tomar medidas para garantizar que el proyecto esté debidamente documentado y analizado.

Cuatro pasos:

- Tener en claro el procesamiento deseado
- Verificar la factibilidad e implementación del programa
- Documentar el análisis
- Realizarlo

Plan:

Estimar cuánto tiempo durará la tarea y decidir qué características se implementarán en qué orden, estableciendo hitos para la finalización de varias partes de las fechas específicas y poner fecha límite final. Esta etapa tiende a estar entrelazada con la etapa de Análisis y Diseño. Puede desarrollar una buena estimación de cuánto tiempo durará la tarea hasta que haya pensado un poco en cómo va a resolverla. Por otro lado, una de las cosas para las que necesita y no tiene tiempo es su propio análisis.

1.4.2 Análisis del problema.

Es la comprensión afondo del problema y sus detalles y es un requisito para lograr una solución eficaz.

1.4.3 Diseño y elaboración del algoritmo.

Dos formas comunes de diseñar la solución a un problema son dibujar un diagrama de flujo y escribir un pseudocódigo, o posiblemente ambos. Esencialmente, un diagrama de flujo es una representación gráfica de una solución paso a paso a un problema. Consiste en flechas que representan la dirección que toma el programa y cuadros y otros símbolos que representan acciones. Es un mapa de lo que su programa va a hacer y cómo lo va a hacer.

El American National Standards Institute (ANSI) ha desarrollado un conjunto estándar de símbolos de diagrama de flujo. La Figura muestra los símbolos y cómo se pueden usar en un diagrama de flujo simple de un acto cotidiano común: preparar una carta para enviarla por correo.

Elseudocódigo es un lenguaje no estándar similar al inglés que le permite establecer su solución con más precisión de la que puede en inglés simple, pero con menos precisión que la que se requiere cuando se utiliza un lenguaje de programación formal. El pseudocódigo le permite concentrarse en la lógica del programa sin tener que preocuparse por la sintaxis precisa de un lenguaje de programación en particular. Sin embargo, el pseudocódigo no es ejecutable en la computadora.

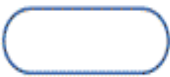
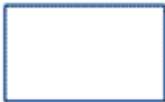


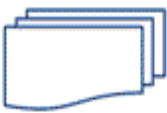





SÍMBOLO	SIGNIFICADO	SÍMBOLO	SIGNIFICADO
	Terminal: Indica el inicio o la terminación del flujo del proceso.		Actividad: Representa la actividad llevada a cabo en el proceso.
	Decisión: Señala un punto en el flujo donde se produce una bifurcación del tipo "Sí" – "No".		Documento: Documento utilizado en el proceso.
	Multidocumento: Refiere un conjunto de documentos. Por ejemplo, un expediente.		Inspección / Firma: Aplicado en aquellas acciones que requieren de supervisión.
	Conector de un Proceso: Conexión o enlace con otro proceso, en el que continúa el diagrama de flujo. Por ejemplo, un subproceso.		Archivo: Se utiliza para reflejar la acción de archivo de un documento o expediente.
	Base de Datos: Empleado para representar la grabación de datos.		Línea de Flujo: Indica el sentido del flujo del proceso.

Figura 4. Diagrama de flujo. Fuente: Recuperado de <https://www.google.com/url?sa=www.aiteco.com%Fdiagrama-de-flujo>

1.4.4 Codificación.

Como programador, su próximo paso es codificar el programa, es decir, expresar su solución en un lenguaje de programación. Traducirás la lógica del diagrama de flujo o pseudocódigo, o alguna otra herramienta, a un lenguaje de programación. Como ya hemos señalado, un lenguaje de programación es un conjunto de reglas que proporciona una forma de instruir a la computadora qué operaciones realizar. Hay muchos lenguajes de programación: BASIC, COBOL, VISUAL STUDIO, JAVA y C son algunos ejemplos. Puede encontrarse trabajando con uno o más de estos.

Aunque los lenguajes de programación operan gramaticalmente, algo así como el idioma inglés, son mucho más precisos. Para que su programa funcione, debe seguir exactamente las reglas, la sintaxis, del idioma que está utilizando. Por supuesto, usar el idioma correctamente no garantiza que su programa funcionará, más que hablar inglés correcto gramaticalmente significa que sabe de lo que está hablando. El punto es que el uso adecuado del lenguaje es el primer paso requerido. Luego, su programa codificado debe estar codificado, probablemente utilizando una terminal o computadora personal, de una forma que la computadora pueda entender.

Una nota más aquí: los programadores generalmente usan un editor de texto, que es algo así como un programa de procesamiento de texto, para crear un archivo que contiene el programa. Sin embargo, como principiante, es probable que primero desee escribir su código de programa en papel.

1.4.5 Prueba y depuración (puesta a punto o testing).

Algunos expertos insisten en que un programa bien diseñado se puede escribir correctamente la primera vez. De hecho, afirman que hay formas matemáticas de demostrar que un programa es correcto. Sin embargo, las imperfecciones del mundo

todavía están con nosotros, por lo que la mayoría de los programadores se acostumbran a la idea de que sus programas recién escritos probablemente tengan algunos errores. Esto es un poco desalentador al principio, ya que los programadores tienden a ser personas precisas, cuidadosas y orientadas a los detalles que se enorgullecen de su trabajo. Aun así, hay muchas oportunidades para introducir errores en los programas, y usted, como aquellos que lo han precedido, probablemente encontrará varios de ellos.

Finalmente, después de codificar el programa, debe prepararse para probarlo en la computadora. Este paso implica estas fases:

Escritorio de cheques. Esta fase, similar a la corrección de pruebas, a veces es evitada por el programador que está buscando un acceso directo y está ansioso por ejecutar el programa en la computadora una vez que está escrito. Sin embargo, con una cuidadosa verificación de escritorio, puede descubrir varios errores y posiblemente ahorrar tiempo a largo plazo. En la verificación de escritorio, simplemente se sienta y rastrea mentalmente, o verifica, la lógica del programa para tratar de asegurarse de que esté libre de errores y sea viable. Muchas organizaciones llevan esta fase un paso más allá con un tutorial, un proceso en el que un grupo de programadores, sus pares, revisan su programa y ofrecen sugerencias de manera colegiada.

Traduciendo. Un traductor es un programa que verifica la sintaxis de su programa para asegurarse de que el lenguaje de programación se utilizó correctamente, proporcionándole todos los mensajes de error de sintaxis, llamados diagnósticos, y luego traduce su programa a una forma de computadora puede entender, un subproducto del proceso es que el traductor le dice si ha utilizado incorrectamente el lenguaje de programación de alguna manera. Este tipo de errores se denominan errores de sintaxis. El traductor produce mensajes de error descriptivos. Por ejemplo, si en FORTRAN escribe erróneamente $N = 2 * (I + J))$, que tiene dos paréntesis de cierre en lugar de uno, recibirá

un mensaje que dice: "Padres Inigualables" (los diferentes traductores pueden proporcionar una redacción diferente para los mensajes de error). Los programas son traducidos más comúnmente por un compilador. La traducción involucra su programa original, llamado módulo fuente, que un compilador transforma en un módulo objeto. Los programas pre escritos de una biblioteca del sistema pueden agregarse durante la fase de enlace / carga, lo que da como resultado un módulo de carga. El módulo de carga puede ser ejecutado por la computadora.

Depuración. Un término usado ampliamente en programación, depuración significa detectar, localizar y corregir errores (errores), generalmente ejecutando el programa. Estos errores son errores lógicos, como decirle a una computadora que repita una operación, pero no decirle cómo dejar de repetir. En esta fase, ejecuta el programa utilizando los datos de prueba que diseña. Debe planificar los datos de la prueba cuidadosamente para asegurarse de probar cada parte del programa.

1.4.6 Documentación.

Documentar es un proceso continuo y necesario, sin embargo, como muchos programadores lo están, puede estar ansioso por realizar actividades más emocionantes centradas en la computadora. La documentación es una descripción detallada por escrito del ciclo de programación y hechos específicos sobre el programa. Los materiales típicos de documentación del programa incluyen el origen y la naturaleza del problema, una breve descripción narrativa del programa, herramientas lógicas como diagramas de flujo y pseudocódigos, descripciones de registros de datos, listas de programas y resultados de pruebas. Los comentarios en el programa en sí también se consideran una parte esencial de la documentación. Muchos programadores documentan mientras codifican. En un sentido

más amplio, la documentación del programa puede ser parte de la documentación de un sistema completo.

El programador inteligente continúa documentando el programa a lo largo de su diseño, desarrollo y prueba. Se necesita documentación para complementar la memoria humana y para ayudar a organizar la planificación del programa. Además, la documentación es crítica para comunicarse con otras personas interesadas en el programa, especialmente otros programadores que pueden ser parte de un equipo de programación. Y, dado que la rotación de personal es alta en la industria de la informática, se necesita documentación escrita para que aquellos que vienen después de usted puedan realizar las modificaciones necesarias en el programa o localizar cualquier error que haya omitido.

1.4.7 Implementación (producción).

Para Trembay (2011) citó al respecto:

El programa ya probado, revisado y mejorado se considera terminado y puede utilizarse con un alto grado de confianza para resolver los problemas que dieron origen a su creación. Si se está automatizando alguna tarea manual, ésta última se desecha para emplear solamente el programa (p.48).

1.4.8 Mantenimiento.

Es la fase de mayor duración, siempre y cuando el programa funcione bien, este debe ser revisado cada cierto tiempo para realizar ajustes si es necesario.

Capítulo II

Herramientas de programación

2.1 Algoritmo

Para realizar un programa bajo un lenguaje de programación determinado, debemos tener herramientas para realizar un programa que no tenga errores. El proceso de diseñar un programa es esencialmente un proceso creativo, ya que un programa se deriva de un problema que queremos resolver, debemos de conocer las etapas que son comunes.

Un algoritmo es una técnica para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

-Preciso: cada orden o instrucción tiene que tener claridad

-Definido: si se repite más de una vez los resultados deben ser iguales.

-Finito: siempre tiene un inicio y un fin.

Crear un algoritmo significa orientar la solución de un problema, de modo que éste pueda ser resuelto con la ayuda de un computador. Entonces diremos que un algoritmo es un conjunto de pasos, instrucciones que se dan para realizar una tarea.

Por ejemplo, si consideramos el algoritmo “El primer día de clases de un alumno de la UNE”, entonces este joven universitario se despierta a las 5 am y seguirá los siguientes pasos:

- Apagar el despertador.

- Levantarse de la cama.
- Asearse.
- Vestirse.
- Organizar sus útiles a llevar.
- Dirigirse al paradero del ómnibus.
- Hacer su cola.
- Subir al ómnibus.
- Bajar del ómnibus.
- Ingresar a la Universidad.
- Buscar el comedor.
- Hacer su cola en el comedor
- Ingresar al comedor.
- Desayunar.
- Buscar el aula.
- Ingresar al aula.

Según Joyanes (2010):

Nos damos cuenta que el joven siguió pasos precisos, definidos y finitos, si cambiaríamos el orden, la respuesta ante eso sería que no llegaríamos a un mismo fin, pero si agregamos más pasos que no alteren el orden, sería otro algoritmo válido también ya que llegaríamos al mismo fin, de ingresar al aula. Las herramientas que mayormente se usan son: pseudocódigos y diagramas de flujo (p.38).

Para llegar al resultado que queremos alcanzar debemos seguir la secuencia de pasos precisos, de esta manera formamos pseudocódigos y diagramas de flujo.

2.2 Pseudocódigo

Joyanes (2010) lo definió:

El pseudocódigo es un lenguaje informal, es una mezcla de lenguaje de programación y español (o cualquier otro idioma), que ayuda a los programadores a desarrollar algoritmos. El pseudocódigo es conveniente y sencillo, se emplea en la programación estructurada para realizar el diseño de un programa, aunque no es un lenguaje de programación real (p.35).

Los programas en pseudocódigos no se ejecutan en las computadoras, sino que solo nos ayudan a “resolver” un programa antes de intentar escribirlo en lenguaje de programación. El pseudocódigo solo consiste en caracteres, de manera que se puede introducir los programas en pseudocódigos a la computadora mediante un programa de edición. Según Joyanes (2010):

La computadora puede desplegar o imprimir una copia reciente del pseudocódigo cuando sea necesario. Un programa en pseudocódigo cuidadosamente preparado puede convertirse fácilmente en su correspondiente programa en muchos casos esto se hace mediante un simple reemplazo de las instrucciones en pseudocódigo por sus equivalentes.

Como ejemplo tenemos a una estructura selectiva simple:

PSEUDOCÓDIGO
Si (condición) entonces Sentencias FinSi

Figura 5. Pseudocódigo ejemplo de estructura selectiva simple. Fuente: Autoría propia.

2.3 Diagrama de flujo

Joyanes (2010) mencionó al respecto:

Es una representación gráfica de un algoritmo. Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI). Aunque su empleo ha disminuido bastante desde la aparición de lenguajes estructurados, es una técnica muy antigua y a la vez más utilizada (p.47).

Esto quiere decir, que el uso de un algoritmo sigue un lineamiento y tiene vigencia a pesar que de la aparición de otras formas de lenguajes estructurados.



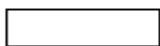






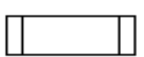




Símbolos principales	Función
	Terminal (representa el comienzo, "inicio", y el final, "fin" de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar un programa).
	Entrada / Salida (cualquier tipo de introducción de datos en la memoria desde los periféricos, "entrada", o registro de la información procesada en un periférico, "salida").
	Proceso (cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etc.)
	Decisión (indica operaciones lógicas o de comparación entre datos-normalmente dos- y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas-respuestas SI o NO- pero puede tener tres o más, según los casos).
	Decisión múltiple (en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).
	Conector (sirve para enlazar dos partes cualesquiera de un organograma a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama).
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones).
	Línea conectora (sirve de unión entre dos símbolos).
	Conector (conexión entre dos puntos del organograma situado en páginas diferentes).
Símbolos secundarios	Función
	Llamada subrutina o a un proceso predeterminado (una subrutina es un módulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa al terminar, al programa principal).
	Pantalla (se utiliza en ocasiones en lugar del símbolo del E/S)
	Impresora (se utiliza en ocasiones en lugar del símbolo del E/S)
	Teclado (se utiliza en ocasiones en lugar de símbolo del E/S)
	Comentarios (se utiliza para añadir comentarios clasificadores a otros símbolos del diagrama de flujo. Se pueden dibujar a cualquier lado del símbolo).

Figura 6. Símbolos del diagrama de flujo. Fuente: Joyanes, 2006.

Capítulo III

Metodología de programación

3.1 Diseño Top Down

El diseño de arriba hacia abajo comienza con una descripción del sistema general y generalmente consiste en una estructura jerárquica que contiene descripciones más detalladas del sistema en cada nivel inferior. Los detalles de diseño de nivel inferior continúan hasta que ya no sea posible una subdivisión adicional, es decir, hasta que el sistema se describa en términos de sus partes "atómicas".

Este método involucra una estructura jerárquica o en forma de árbol para un sistema.

3.2 Programación modular

Se refiere al proceso de subdividir un programa de computadora en subprogramas separados. Un módulo es un componente de software separado. A menudo se puede usar en una variedad de aplicaciones y funciones con otros componentes del sistema. Se agrupan funciones similares en la misma unidad de código de programación y se desarrollan funciones separadas como unidades de código separadas para que otras aplicaciones puedan reutilizar el código.

La programación orientada a objetos (OOP) es compatible con el concepto de programación modular en gran medida. La programación modular permite a múltiples programadores dividir el trabajo y depurar partes del programa de forma independiente.

Techopedia explica la programación modular. Los módulos en programación modular imponen límites lógicos entre componentes y mejoran la capacidad de mantenimiento. Se incorporan a través de interfaces. Están diseñados de tal manera que minimizan las dependencias entre los diferentes módulos. Los equipos pueden desarrollar módulos por separado y no requieren el conocimiento de todos los módulos del sistema.

Los módulos en programación modular imponen límites lógicos entre componentes y mejoran la capacidad de mantenimiento. Se incorporan a través de interfaces. Están diseñados de tal manera que minimizan las dependencias entre los diferentes módulos. Los equipos pueden desarrollar módulos por separado y no requieren el conocimiento de todos los módulos del sistema.

Todas y cada una de las aplicaciones modulares tienen asociado un número de versión. Esto proporciona flexibilidad a los desarrolladores en el mantenimiento del módulo. Si se debe aplicar algún cambio a un módulo, solo se deben cambiar las subrutinas afectadas. Esto hace que el programa sea más fácil de leer y comprender.

La programación modular tiene un módulo principal y muchos módulos auxiliares. El módulo principal se compila como un ejecutable (EXE), que llama a las funciones del módulo auxiliar. Los módulos auxiliares existen como archivos ejecutables separados, que se cargan cuando se ejecuta el EXE principal. Cada módulo tiene un nombre único asignado en la instrucción PROGRAM. Los nombres de las funciones en los módulos deben ser únicos para facilitar el acceso si las funciones utilizadas por el módulo principal deben exportarse.

Los idiomas que admiten el concepto de módulo son IBM Assembler, COBOL, RPG, FORTRAN, Morpho, Zonnon y Erlang, entre otros.

Los beneficios de usar programación modular incluyen:

- Menos código tiene que ser escrito.
- Se puede desarrollar un único procedimiento para su reutilización, eliminando la necesidad de volver a escribir el código muchas veces.
- Los programas se pueden diseñar más fácilmente porque un equipo pequeño solo se ocupa de una pequeña parte de todo el código.
- La programación modular permite a muchos programadores colaborar en la misma aplicación.
- El código se almacena en varios archivos.
- El código es corto, simple y fácil de entender.
- Los errores se pueden identificar fácilmente, ya que se localizan en una subrutina o función.
- El mismo código se puede usar en muchas aplicaciones.
- El alcance de las variables se puede controlar fácilmente.

3.3 Programación estructurada

La programación estructurada es un subconjunto de programación procesal que reduce la necesidad de declaraciones goto. En muchos sentidos, OOP se considera un tipo de programación estructurada que implementa técnicas de programación estructurada. Ciertos lenguajes, como Pascal, Algorithmic Language (ALGOL) y Ada, están diseñados para imponer una programación estructurada.

El concepto de programación estructurada fue formalizado en 1966 por Corrado Böhm y Giuseppe Jacopini, quienes demostraron el diseño teórico de programas de

computadora a través de bucles, secuencias y decisiones. A fines de la década de 1960 y principios de la de 1970, Edsger W.Dijkstra desarrolló la funcionalidad de programación estructural como un método ampliamente utilizado, en el que un programa se divide en múltiples secciones con múltiples salidas y un punto de acceso.

La programación estructurada es un paradigma de programación destinado a mejorar la claridad, la calidad y el tiempo de desarrollo de un programa informático mediante el uso extensivo de las estructuras de flujo de control estructurado de selección (si / luego / más) y repetición (mientras y para), estructuras de bloques , y subrutinas en contraste con el uso de pruebas y saltos simples como la declaración go to, que puede conducir a un "código de espagueti" que es potencialmente difícil de seguir y mantener.

Discusión

Uno de los conceptos más importantes de la programación es la capacidad de controlar un programa para que se ejecuten diferentes líneas de código o que algunas líneas de código se ejecuten muchas veces. Los mecanismos que nos permiten controlar el flujo de ejecución se denominan estructuras de control. El diagrama de flujo es un método para documentar (trazar) el flujo (o rutas) que un programa ejecutaría. Hay tres categorías principales de estructuras de control:

- **Secuencia** - Muy aburrida. Simplemente haga una instrucción, luego la siguiente y la siguiente. Simplemente hazlos en una secuencia dada o en el orden indicado. La mayoría de las líneas de código son esto.
- **Selección:** aquí es donde selecciona o elige entre dos o más flujos. La elección se decide haciendo algún tipo de pregunta. La respuesta determina la ruta (o qué líneas de código) se ejecutará.

- **Iteración:** también conocida como repetición, permite ejecutar (o repetir) un código (de una a muchas líneas) varias veces. Es posible que el código no se ejecute en absoluto (repítalo cero veces), se ejecute un número fijo de veces o se ejecute indefinidamente hasta que se cumpla alguna condición. También conocido como bucle porque el diagrama de flujo muestra el flujo en bucle para repetir la tarea.

Una cuarta categoría describe el código no estructurado.

- **Ramificación:** una estructura no controlada que permite que el flujo de ejecución salte a una parte diferente del programa. Esta categoría rara vez se usa en la programación estructurada modular.

Todos los lenguajes de programación de alto nivel tienen estructuras de control. Todos los lenguajes tienen las tres primeras categorías de estructuras de control (secuencia, selección e iteración). La mayoría tiene la estructura if then else (que pertenece a la categoría de selección) y la estructura while (que pertenece a la categoría de iteración). Después de estas dos estructuras básicas, generalmente hay variaciones de lenguaje.

El concepto de programación estructurada comenzó a fines de la década de 1960 con un artículo de Edsger Dijkstra. Propuso un método de "ir a menos" para planificar la lógica de programación que eliminó la necesidad de la categoría de ramificación de las estructuras de control. El tema fue debatido durante unos 20 años. Pero en última instancia: "A fines del siglo XX, casi todos los informáticos estaban convencidos de que es útil aprender y aplicar los conceptos de programación estructurada.

3.4 Programación no estructurada

La programación no estructurada es el paradigma de programación más antiguo capaz de crear algoritmos completos. A esta programación le sigue la programación por

procedimientos y luego la programación orientada a objetos, ambas consideradas programaciones estructuradas.

Según Joyanes (2010) “la programación no estructurada ha sido criticada por producir código difícil de seguir, llamado despectivamente de libre caída. Aunque sí ha sido elogiada por la libertad que ofrece a los programadores” (p.54).

Existen lenguajes de programación de alto y bajo nivel que utilizaban programación no estructurada. Algunos ejemplos son: las primeras versiones de BASIC (como MSX BASIC y GW-BASIC), FORTRAN, JOSS, FOCAL, MUMPS, TELCOMP, COBOL, códigos a nivel máquina, primeros sistemas assembler (sin metaoperadores procedurales), depuradores assembler y algunos lenguajes scripting como los archivos batch de MS-DOS.

3.5 Programación orientada a objetos (POO)

Programación orientada a objetos: como su nombre lo indica, la programación orientada a objetos u OOP se refiere a lenguajes que usan objetos en la programación. La programación orientada a objetos tiene como objetivo implementar entidades del mundo real como herencia, ocultación, polimorfismo, etc. en la programación. El objetivo principal de OOP es unir los datos y las funciones que operan en ellos para que ninguna otra parte del código pueda acceder a estos datos, excepto esa función.

Conceptos de OOP: polimorfismo, herencia, encapsulamiento, abstracción, clase, objeto, método y paso de mensajes

Aprendamos sobre las diferentes características de un lenguaje de programación orientado a objetos:

- **Polimorfismo.** El polimorfismo se refiere a la capacidad de los lenguajes de programación OOP para diferenciar entre entidades con el mismo nombre de manera eficiente. Java lo hace con la ayuda de la firma y declaración de estas entidades.

- **Herencia.** La herencia es un pilar importante de la OOP (Programación Orientada a Objetos). Es el mecanismo en Java por el cual una clase puede heredar las características (campos y métodos) de otra clase. Podemos mencionar la siguiente terminología importante:
 - Superclase: La clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal).
 - Subclase: La clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase secundaria). La subclase puede agregar sus propios campos y métodos además de los campos y métodos de la superclase.
 - Reusabilidad: la herencia admite el concepto de "reusabilidad", es decir, cuando queremos crear una nueva clase y ya hay una clase que incluye parte del código que queremos, podemos derivar nuestra nueva clase de la clase existente. Al hacer esto, estamos reutilizando los campos y métodos de la clase existente.

La palabra clave utilizada para la herencia se extiende.

- **Encapsulamiento.** La encapsulación se define como la recopilación de datos en una sola unidad. Es el mecanismo que une el código y los datos que manipula. Otra forma de pensar en la encapsulación es que es un escudo protector que evita que el código acceda a los datos fuera de este escudo.

Técnicamente en la encapsulación, las variables o datos de una clase están ocultos de cualquier otra clase y solo se puede acceder a ellos a través de cualquier función miembro de la propia clase en la que se declaran. Al igual que en la encapsulación, los datos de una clase están ocultos de otras clases, por lo que también se conoce como ocultación de datos.

La encapsulación se puede lograr declarando todas las variables de la clase como privadas y escribiendo métodos públicos en la clase para establecer y obtener los valores de las variables.

- **Abstracción.** La abstracción de datos es la propiedad en virtud de la cual solo los detalles esenciales se muestran al usuario. Las unidades triviales o no esenciales no se muestran al usuario. Ej: Un automóvil se ve como un automóvil en lugar de sus componentes individuales.

La abstracción de datos también se puede definir como el proceso de identificar solo las características requeridas de un objeto ignorando los detalles irrelevantes. Las propiedades y los comportamientos de un objeto lo diferencian de otros objetos de tipo similar y también ayudan a clasificar / agrupar los objetos.

Considere un ejemplo de la vida real de un hombre conduciendo un automóvil. El hombre solo sabe que al presionar los aceleradores aumentará la velocidad del automóvil o al aplicar los frenos se detendrá el automóvil, pero no sabe cómo al presionar el acelerador la velocidad realmente aumenta, no sabe sobre el mecanismo interno del automóvil o La implementación de acelerador, frenos, etc. en el automóvil. Esto es lo que es la abstracción.

En java, la abstracción se logra mediante interfaces y clases abstractas. Podemos lograr el 100% de abstracción utilizando interfaces.

- **Clase.** Una clase es un modelo o prototipo definido por el usuario a partir del cual se crean los objetos. Representa el conjunto de propiedades o métodos que son comunes a todos los objetos de un tipo. En general, las declaraciones de clase pueden incluir estos componentes, en orden:

-Modificadores: una clase puede ser pública o tener acceso predeterminado (consulte esto para más detalles).

-Nombre de clase: el nombre debe comenzar con una letra inicial (en mayúscula por convención).

-Superclase (si existe): el nombre del padre de la clase (superclase), si corresponde, precedido por la palabra clave `extends`. Una clase solo puede extender (subclase) un padre.

-Interfaces (si las hay): una lista de interfaces separadas por comas implementadas por la clase, si las hay, precedidas por los implementos de palabras clave. Una clase puede implementar más de una interfaz.

-Cuerpo: el cuerpo de la clase rodeado de llaves, `{ }`.

- **Objeto.** Es una unidad básica de programación orientada a objetos y representa las entidades de la vida real. Un programa Java típico crea muchos objetos que, como saben, interactúan invocando métodos. Un objeto consta de:

-**Estado:** está representado por los atributos de un objeto. También refleja las propiedades de un objeto.

-**Comportamiento:** se representa mediante métodos de un objeto. También refleja la respuesta de un objeto con otros objetos.

-**Identidad:** le da un nombre único a un objeto y permite que un objeto interactúe con otros objetos.

Ejemplo de un objeto: perro

Nombre del Perro: Bobby

Características: Color Marrón. 5 años, Pastor Alemán

Comportamiento: Ladra poco, duerme mucho come regular.

- **Método.** Un método es una colección de declaraciones que realizan alguna tarea específica y devuelven el resultado al llamante. Un método puede realizar una tarea específica sin devolver nada. Los métodos nos permiten reutilizar el código sin volver a escribir el código. En Java, cada método debe ser parte de alguna clase que sea diferente de lenguajes como C, C++ y Python.

Los métodos ahorran tiempo y nos ayudan a reutilizar el código sin tener que volver a escribirlo.

- **Declaración del método**

En general, las declaraciones de métodos tienen seis componentes:

-Modificador de acceso: define el tipo de acceso del método, es decir, desde dónde se puede acceder en su aplicación. En Java, hay 4 tipos de especificadores de acceso.

-public: accesible en toda clase en su aplicación.

-protegido: accesible dentro del paquete en el que está definido y en sus subclases (incluidas las subclases declaradas fuera del paquete)

-privado: accesible solo dentro de la clase en la que se define.

-predeterminado (declarado / definido sin usar ningún modificador): accesible dentro de la misma clase y paquete dentro del cual se define su clase.

-El tipo de retorno: el tipo de datos del valor devuelto por el método o nulo si no devuelve un valor.

-Nombre del método: las reglas para los nombres de campo se aplican también a los nombres de métodos, pero la convención es un poco diferente.

-Lista de parámetros: la lista separada por comas de los parámetros de entrada se define, precedida por su tipo de datos, dentro del paréntesis adjunto. Si no hay parámetros, debe usar paréntesis vacíos ().

-Lista de excepciones: las excepciones que espera por el método pueden arrojar, puede especificar estas excepciones.

-Cuerpo del método: está encerrado entre llaves. El código que debe ejecutarse para realizar sus operaciones previstas.

- **Paso del mensaje.** Los objetos se comunican entre sí enviándose y recibiendo información entre ellos. Un mensaje para un objeto es una solicitud de ejecución de un

procedimiento y, por lo tanto, invocará una función en el objeto receptor que genera los resultados deseados. La transmisión de mensajes implica especificar el nombre del objeto, el nombre de la función y la información que se enviará.

Aplicación didáctica

Sesión de aprendizaje

Título: Definimos un algoritmo y lo ejecutamos en Pseint.

I. DATOS INFORMATIVOS:

Área	Computación e informática	I.E.	N° 1237
Grado	1° de secundaria	Sección	“ A”
Docente	Cáceres Espinoza, Liliana	Fecha	15/05/2019

II. PROPOSITO DE LA SESIÓN:

Los estudiantes definirán que es un algoritmo y los ejecutarán en Pseint.

III. PREPARACIÓN ANTES DE LA SESIÓN:

ANTES DE LA SESION:


- ❖ Preparamos el aula de cómputo, para el trabajo con los estudiantes.
- ❖ Se saca copias en las hojas de información para cada estudiante

MATERIALES O RECURSOS A UTILIZAR

Computadoras
Cuadernos y lapiceros
Hojas de información
Hoja de evaluación

IV. PROPÓSITO DEL APRENDIZAJE:

Competencia/Capacidad	Desempeño	Evidencia	Instrumentos de valoración
Se desenvuelve en entornos virtuales generados por las TIC. <ul style="list-style-type: none"> • Personaliza entornos virtuales • Gestiona información del entorno virtual. • Interactúa en entornos virtuales. • Crea objetos virtuales en diversos formatos. 	Desarrolla procedimientos lógicos y secuenciales para plantear soluciones a enunciados concretos con lenguajes de programación.	<ul style="list-style-type: none"> • Elabora un diagrama de flujo, para resolver un problema. • Utilizan Pseint. 	<ul style="list-style-type: none"> • Lista de cotejo • Fichas de aplicación • Cuaderno de trabajo

MOMENTOS DE LA SESIÓN	
Inicio:	Tiempo aproximado: 10 minutos
<p>Grupo de clase:</p> <ul style="list-style-type: none"> ➤ Saluda amablemente a los estudiantes e inicia la sesión comentando lo trabajado en la sesión anterior. ➤ Se pregunta: ¿Qué trabajamos en la clase anterior? ¿Qué era un algoritmo? ➤ Se escuchan sus ideas. ➤ Se comenta un caso: De realizar una suma usando algoritmos. Se pregunta: ¿Qué deben hacer los estudiantes para sumar? ➤ Se escuchan sus ideas. <p>Comunica el propósito de la sesión:</p> <div style="display: flex; align-items: center;">  <div style="border: 2px solid red; border-radius: 15px; padding: 10px; margin-left: 20px; background-color: #e6f2ff;"> <p style="text-align: center;">Definimos un algoritmo y lo ejecutamos en Pseint</p> </div> </div> <ul style="list-style-type: none"> ➤ Se acuerda con los estudiantes algunas normas de convivencia que los ayudarán a trabajar mejor. <div style="border: 2px solid yellow; border-radius: 15px; padding: 10px; margin-top: 20px; background-color: #ffff00;"> <p style="text-align: center;">Normas de convivencia</p> <ul style="list-style-type: none"> ➤ Guardar silencio en la clase. ➤ Escuchar y respetar a mis compañeros. ➤ Trabajar en equipo sin molestar. </div>	
Desarrollo	Tiempo aproximado: 65 minutos
<ul style="list-style-type: none"> ➤ Terminado, se realizara un resumen, por los alumnos, planteando elaborar algoritmos en busca de soluciones de problemas. ➤ Se entrega muestra a los estudiantes información sobre algoritmo, a través del multimedia, se pide que lo lean, y sacamos conclusiones de ello, definiendo así un algoritmo. ➤ Se cierra la información con una explicación previa, por parte del docente. ➤ Seguidamente, recordamos el propósito y el caso mencionado en el inicio, se pregunta: ¿podemos usar algoritmos para solucionar ese caso? ¿cómo? se escuchan sus ideas . <p>Trabajo individual</p> <ul style="list-style-type: none"> ➤ Se pide a los estudiantes que, en sus PC, abran el programa Pseint, se explica brevemente la función del programa. ➤ Se pide a los estudiantes que ejecuten un algoritmo para crear sumas. ➤ Se entrega a los estudiantes una hoja de información con los pasos. ➤ Se pasará por cada estudiante, disipando dudas y apoyándolos en todo momento. ➤ Al finalizar se pide a un voluntario, explicar lo trabajado en clase. ➤ Se evaluará el proceso mediante una lista de cotejo. 	

CIERRE	10 Minutos
<ul style="list-style-type: none"> ➤ Se verifica que todos han concluido su trabajo. ➤ Se propicia la metacognición con las siguientes preguntas: ¿qué hemos hecho hoy?, ¿qué se consideró para la elaboración?, ¿qué es lo que más les gusta de su trabajo? ➤ Se retoma el propósito y pregunta: ¿Qué es un algoritmo? ¿Cómo lo ejecutamos? 	
Tarea para la casa: <ul style="list-style-type: none"> ➤ Se pedirá a los estudiantes, que escriban en sus cuadernos los pasos seguidos en la clase de hoy. ➤ Se pide ejecutar una multiplicación usando en Pseint. 	

 Directora

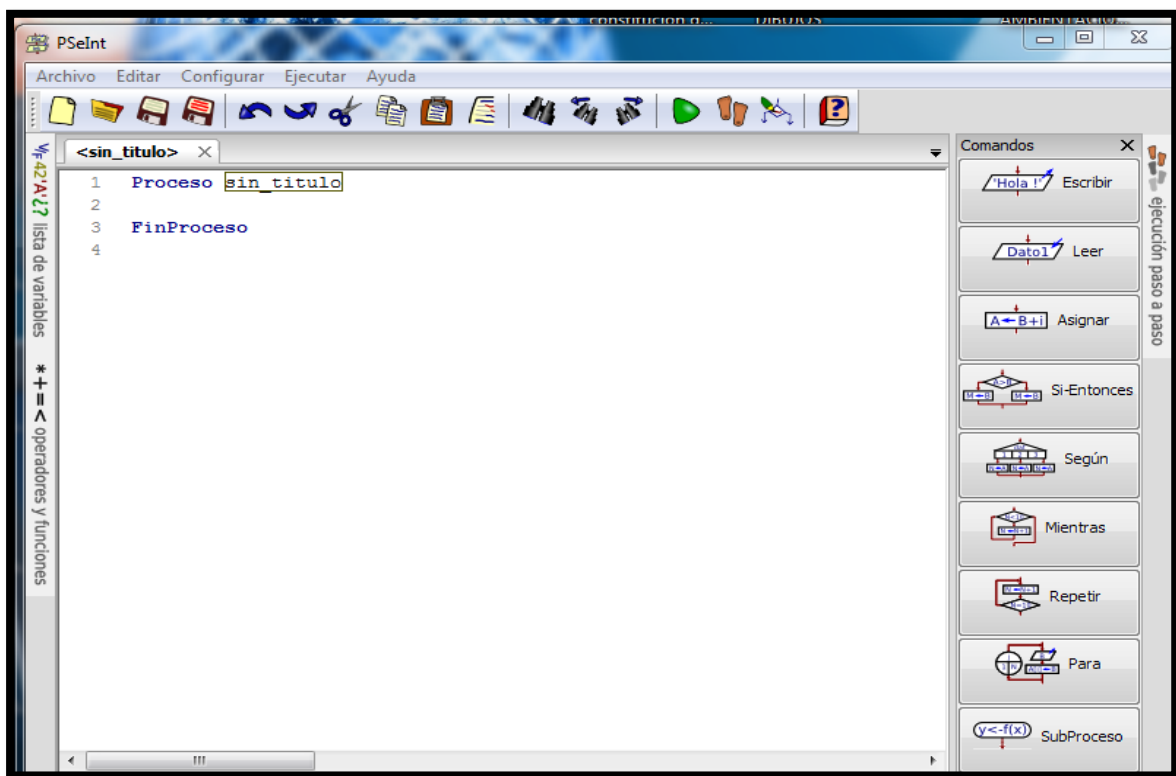
 Profesora

Hoja de información

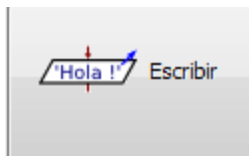
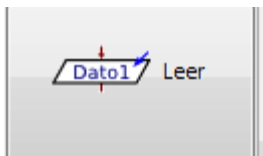
Ejercicio: Realizar con PSEINT para hallar la suma de dos números naturales.

Pasos:

1. Prender la PC.
2. Abrir **PSEINT**
3. Muestra la pantalla:



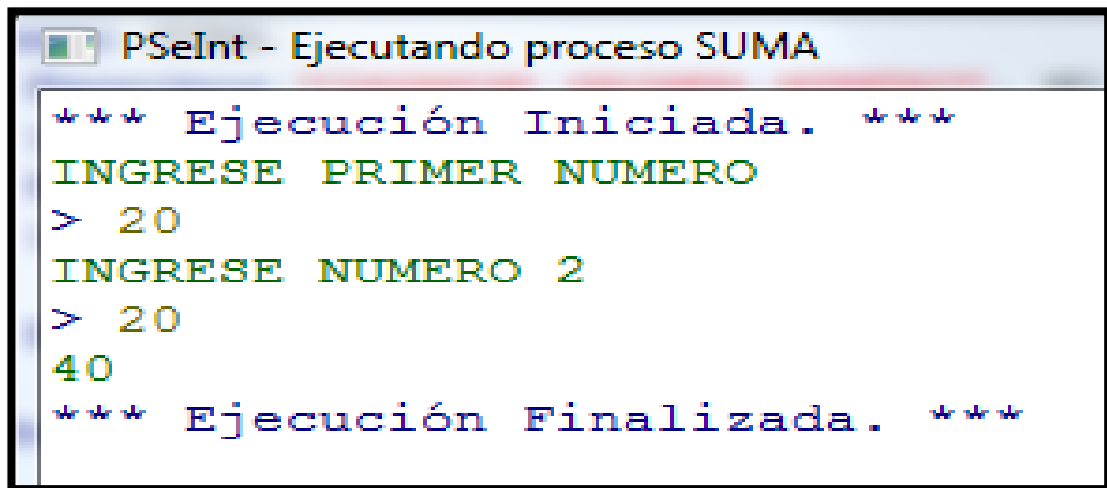
- Insertamos los siguientes comandos e ingresamos los datos.



```

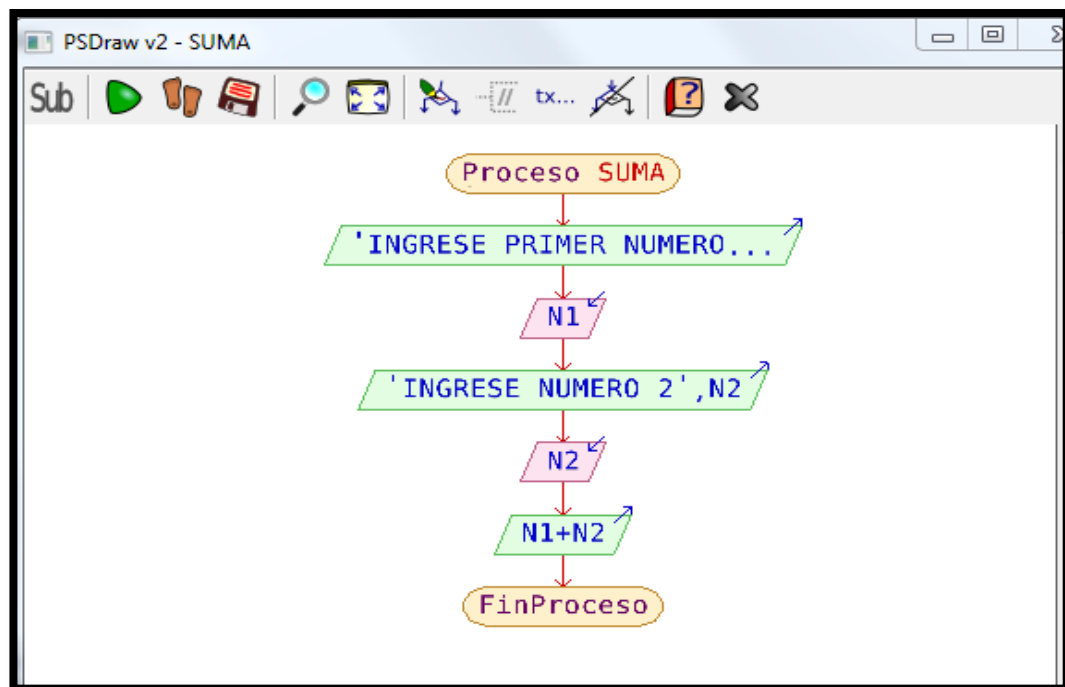
1  Proceso SUMA
2      Escribir "INGRESE PRIMER NUMERO", N1
3      Leer N1
4      Escribir "INGRESE NUMERO 2", N2
5      Leer N2
6
7      Escribir N1+N2
8
9  FinProceso
  
```


4. Ejecutando quedara de la siguiente forma

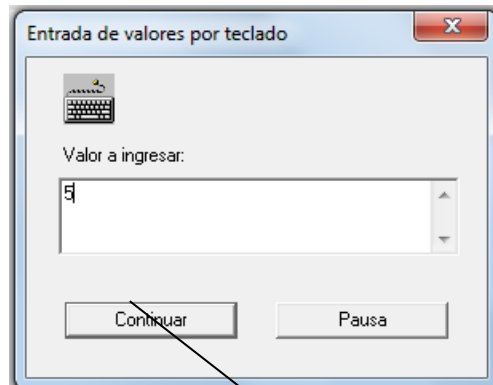


```
*** Ejecución Iniciada. ***  
INGRESE PRIMER NUMERO  
> 20  
INGRESE NUMERO 2  
> 20  
40  
*** Ejecución Finalizada. ***
```

5. En el programa PSEINT ejecutar y visualizar el flujo grama.



6. Ejecute el diagrama y cuando solicite un valor, introduzca 5, posteriormente aparecerá la misma ventana, introduzca el número 7, observe la ilustración siguiente:

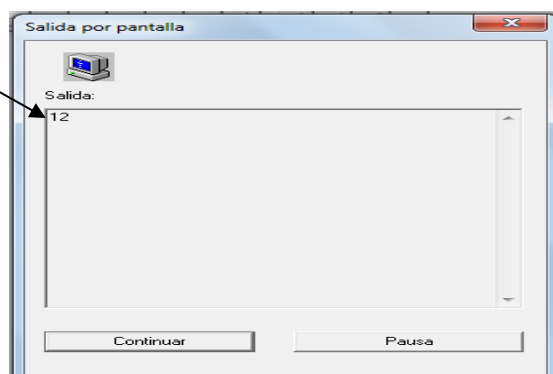


Haga clic en el botón **Continuar**

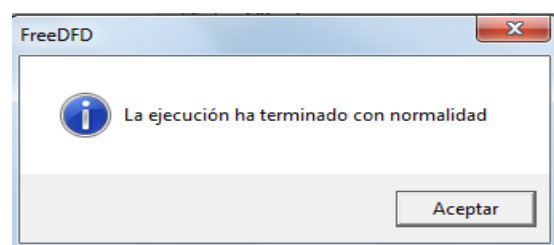
Nota: Si hay errores al ejecutar, se pondrá en color Rojo, corríjalos y vuelva a ejecutar.

7. Los resultados son mostrados:

Resultados de la ejecución.



8. Al finalizar de forma correcta saldrá la siguiente ventana. Revisión por parte del asesor.



Síntesis

Un sistema en general se define como conjunto de componentes concentrados e interactivos, que tienen un propósito y una unidad total. Sistema de procesamiento de información es un sistema que transforma datos brutos en información organizada, significativa y útil, basados en un esquema simple como un proceso productivo donde el ingreso de materia prima equivaldría al Input, luego la transformación de esta sería el procesamiento de datos y por último el producto terminado es el Output, muchos dispositivos u organismos pueden ser considerados sistemas de procesamiento de la información.

La entrada es la temperatura media y la salida es una señal que controla la caldera del aire acondicionado. El corazón de un animal o un ser humano es un sistema complejo de procesamiento de la información.

El conjunto de instrucciones que especifican la secuencia de operaciones a realizar, en orden, para resolver un sistema específico o clase de problemas, se denomina algoritmo. Definimos como algoritmo a un conjunto de pasos necesarios para resolver un problema ya sea manualmente o por métodos mecanizados que, son los más usuales en la actualidad.

El proceso de la creación de software requiere el uso de una metodología sistemática de desarrollo que permita un acercamiento gradual a la solución del problema que se intenta resolver. Esta metodología llamada Ciclo de Desarrollo del Software, consta de una serie de pasos lógicos secuenciales o fases.

Apreciación crítica y sugerencias

La computadora permite procesar datos para generar información, este proceso es en forma automática, para lo cual realiza muchas operaciones. También, podemos diseñar soluciones a la medida, de ciertos problemas que se nos presentan., los cuales pueden involucrar operaciones matemáticas complejas o repetitivas, o volumen muy grande de datos para su proceso.

Actualmente la globalización exige estar preparados en el tema de lenguaje de programación debido a que en el trabajo como en la educación se vienen implementando cada día herramientas tecnológicas de computación con el objetivo de efectivizar las tareas.

Referencias

- Cibertec (2015). *Lenguaje de programación I*. Lima: Universidad Privada de Ciencias Aplicadas S.A.C.
- Carrasco, J. (2012). *Programación en computadoras*. México: Mc Graw-Hill
- Castañeda, O., J., Vargas, G., M. del C., León, C., J. (2013). *Programación con Visual Basic*. Tijuana, México: Revista Iberoamericana de Producción Académico y Gestión Educativa.
- García, B., Jaén, G., J., A., y Martínez, F. R. (2002). *Métodos Informáticos en Turbo Pascal*. Madrid: Bellisco.
- Joyanes, A., L. (2010). *Metodología de la programación*. Madrid: McGraw-Hill.
- Joyanes, L. (2006). *Fundamentos de programación, algoritmos y estructura de datos*. 3ra Edición. Madrid: McGraw-Hill.
- Tremblay, J., P. (2011). *Introducción a las ciencias de las computadoras*. Estados Unidos: McGraw-Hill