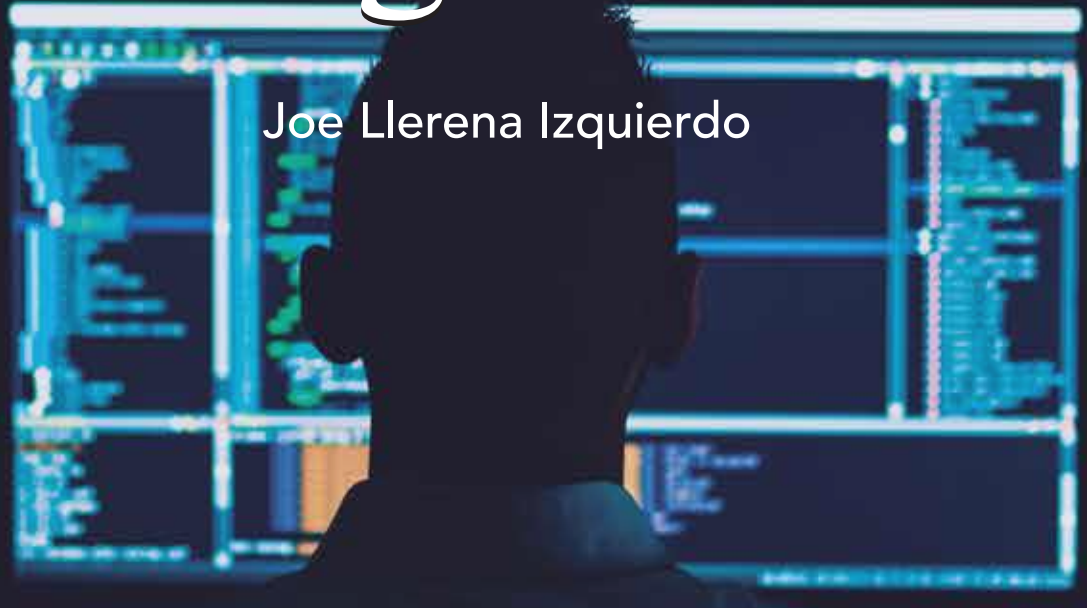
Serie
Te acompaño a programar

Volumen II

Guía de aprendizaje de Programación



Joe Llerena Izquierdo

Universidad Politécnica Salesiana

Guía de aprendizaje de Programación

Serie

Te acompaño a programar

Volumen II

Guía de aprendizaje de Programación

Joe Llerena Izquierdo

ABYA YALA | UPS

2023

GUÍA DE APRENDIZAJE DE PROGRAMACIÓN

1ra edición: Universidad Politécnica Salesiana
Av. Turuhuayco 3-69 y Calle Vieja
Cuenca-Ecuador
Casilla: 2074
P.B.X. (+593 7) 2050000
Fax: (+593 7) 4 088958
e-mail: rpublicas@ups.edu.ec
www.ups.edu.ec

CARRERA DE COMPUTACIÓN

Diagramación: Latex de autor

ISBN
Obra completa: 978-9978-10-428-6

ISBN impreso: 978-9978-10-769-0

ISBN digital: 978-9978-10-770-6

Tiraje: 300 ejemplares

Impreso en Quito-Ecuador, enero 2023

Publicación arbitrada de la Universidad Politécnica Salesiana

Al equipo de Coordinación Académica de la UPS sede Guayaquil.
A la Carrera de Ingeniería en Computación de la UPS sede Guayaquil.
Al Claustro de Computación Aplicada y Metodologías de la Computación.
A los estudiantes de los primeros años de las carreras de Ingenierías.
Universidad Politécnica Salesiana

Bienvenida

Esta guía es fruto del esfuerzo y preocupación de un conjunto de profesores que tomamos la decisión de compartir contigo los conocimientos de la asignatura de Programación. Es el medio de poder acompañarte. ¡Es para ti!, que como otros estudiantes y debido a las condiciones distintas que encontramos abre la posibilidad de comunicarnos por medio de este documento, dejando plasmado un conjunto de actividades que podrán darte una guía en este estudio.

Esperamos que esta Guía de Aprendizaje, elaborada con el propósito de que puedas desarrollar tu aprendizaje y estudio, minimice la necesidad y dependencia de la Internet, y te permite un aprendizaje autorregulado.

Cordialmente,

Tu profesor

Contenido

Índice de Figuras

Índice de Tablas

Índice de Desafíos

Índice de Algoritmos

Índice de Códigos Fuente

Datos Informativos de la Asignatura

PROGRAMA ANALÍTICO DE LA ASIGNATURA

1. Datos informativos

Carrera:	**COMPUTACIÓN [AJUSTE CURRICULAR 2019]**		
Asignatura:	Programación	**Nivel:**	1
Código de la Asignatura:	C-CT-ICO-101	**Unidad de Organización Curricular:**	
Número total de horas:	160	UNIDAD BÁSICA	
N° Horas Componente Docencia:	64	**Campo de Formación:**	
N° Horas Componente Prácticas de Aplicación y Experimentación de Aprendizajes:	32	FUNDAMENTOS TEÓRICOS	
N° Horas Componente de Trabajo Autónomo:	64	**Modalidad:**	PRESENCIAL

2. Caracterización de la Asignatura

Dentro de la ingeniería, es necesario el análisis, diseño e implementación de una solución a un problema planteado de forma algorítmica. La asignatura de programación permite la integración de técnicas de solución de problemas, el uso de algoritmos computacionales y los conocimientos de lenguajes de programación.

3. Resultados de Aprendizaje
   - Desarrolla aplicaciones utilizando el lenguaje de programación.
   - Diseña algoritmos para la resolución de problemas.
   - Diseña soluciones a problemas de lógica empleando álgebra de Boole y teoría de proposiciones.

4. Contenidos

Unidades Temáticas	Contenidos de la Unidad	Resultados de Aprendizaje de la Asignatura correspondientes a cada	Indicadores de Logro	Total de horas por unidad
UNIDAD 1 Lógica Matemática	Álgebra de Boole Teoría de proposiciones	Diseña soluciones a problemas de lógica empleando álgebra de Boole y teoría de proposiciones.	*Aplica lógica matemática para resolver problemas.*	30
UNIDAD 2 Algoritmos	Formas de representar algoritmos Estructuras de Control (Decisión, Repetitivas)	Diseña algoritmos para la resolución de problemas.	*Utiliza estructuras de control para resolver problemas a través de algoritmos.*	50
UNIDAD 3 Lenguaje de Programación	Fundamentos (generalidades, tipos de datos, operadores, expresiones, variables) Funciones Vectores, Matrices	Desarrolla aplicaciones utilizando el lenguaje de programación.	*Identifica la sintaxis y semántica en un lenguaje de programación.* *Diseña e implementa programas en un lenguaje de programación.*	80

5. Metodologías de aprendizaje

El Aprendizaje colaborativo, estrategia para incentivar el diálogo y discusión entre los estudiantes y con el docente, donde todos los miembros colaboran en la construcción del conocimiento y contribuyen al aprendizaje de todos.

Aprendizaje basado en problemas, mediante esta metodología, los estudiantes deben encontrar una solución a un problema planteado, de este modo consiguen elaborar un diagnóstico de las necesidades de aprendizaje, construir el conocimiento y trabajar cooperativamente.

Metodología “aula invertida”, es una metodología de aprendizaje semipresencial, donde los estudiantes aprenden los conceptos desde casa viendo videos en línea y los ejercicios que anteriormente fueron realizados en clase.

El Aprendizaje basado en investigación, considera al estudiante como protagonista de su propio aprendizaje. Los estudiantes actúan como investigadores, aprenden y desarrollan habilidades investigativas mediante la experimentación. La enseñanza se orienta a ayudar a los estudiantes a comprender los fenómenos de la forma en que lo hacen los expertos.

Aprendizaje basado en proyectos.
Prácticas de laboratorio: propende a que los estudiantes adquieran las habilidades propias de los métodos de la investigación científica, amplíen, profundicen, consoliden, realicen y comprueben los fundamentos de la asignatura mediante experimentación.

6. Procedimiento de Evaluación

Según el Reglamento Interno de Régimen Académico:

Artículo 41.- Evaluación de aprendizajes. - Para la aprobación de asignaturas en los niveles de grado, independientemente de la modalidad de estudios, el estudiante debe demostrar do-

minio de conocimientos, capacidades, destrezas y desempeños previstos en los resultados de aprendizaje. La evaluación se realiza en forma sistemática y continua sobre un total de cien puntos divididos en dos partes de cincuenta puntos cada una, que incluyen aprovechamiento y examen. La nota mínima para la aprobación es de setenta puntos.

Artículo 42.- El aprovechamiento será evaluado y calificado con un mínimo de treinta puntos, considerando los resultados de aprendizaje previstos en la planificación micro curricular y las actividades de aprendizaje desarrolladas. La calificación de aprovechamiento será el resultado de por lo menos tres actividades de aprendizaje, sean éstas de carácter colaborativo, prácticas de aplicación y experimentación, trabajo autónomo, u otras:

De carácter colaborativo:

(a). Sistematización de prácticas de investigación-intervención,
(b). Proyectos de integración de saberes,
(c). Construcción de modelos y prototipos,
(d). Proyectos de problematización,
(e). Resolución de problemas o casos.

De prácticas de aplicación y experimentación:

(a). Prácticas de campo,
(b). Trabajos de observación dirigida,
(c). Resolución de problemas,
(d). Talleres.

De trabajo autónomo:

(a). Elaboración individual de ensayos,
(b). Trabajos y exposiciones,
(c). Pruebas orales o escritas,
(d). Resolución de guías didácticas,
(e). Indagación bibliográfica.

7. Bibliografía

TEXTOS BÁSICOS	AUTOR; TÍTULO; EDICIÓN Y AÑO
1	L. Joyanes Aguilar; Fundamentos de programación Algoritmos y Estructuras de Datos y Objetos; México: McGraw-Hill; 2013
2	Thomas H. Cormen; Introduction to Algorithms; Third edition; MIT Press; 2009
3	Steven S. Skiena; The algorithm Design Manual; Second edition; Springer; USA, 2008
4	Kent D. Lee; Foundations of Programming Languages; Springer; 2014
5	Arvind Kumar Bansal; Introduction to Programming Languages; CRC Press, 2014
6	Holly Moore; Matlab para Ingenieros; Pearson - Prentice Hall; 2007

LECTURAS SUGERIDAS	
1	Rod Stephens, Essential Algoritms (A Practical Approach to Computer Algorithms), Wiley, 2013
2	Mauricio Ortiz, Andrea Plaza, Fundamentos de Programación en JAVA y UML, UPS, Cuenca, 2013
3	Llerena Izquierdo, Joe, Codifica en Python, 2020

Sugerencias antes de iniciar su trabajo académico:

Comparto a usted algunas recomendaciones que pueden ayudar a ser un estudiante universitario responsable y sobre todo ejemplo para su familia, compañeros y sociedad en general, a través de la vivencia de valores personales entre otros como responsabilidad, dedicación, constancia, transparencia.

a) Establezca el tiempo y espacio físico adecuados en su agenda diaria para la realización de las tareas académicas contempladas en la guía de aprendizaje.

b) Al realizar las tareas académicas, dedique su pensamiento y acciones únicamente a las referidas a la tarea, esforzándose en abstraerse de las distracciones que todos tenemos y así su trabajo académico será provechoso, optimizando al máximo su tiempo, dedicación, la calidad estará agregada en su trabajo académico.

c) Lea con detenimiento y sin prisa, el contenido de la bibliografía haga notas, subraye, resalte, genere esquemas, mapas conceptuales o lo que para usted es más fácil, sobre lo que le llame más la atención en la lectura, identificando el mensaje o idea principal.

d) Lea nuevamente la Guía de Aprendizaje y centre su atención en lo que se le pide en el trabajo académico.

e) Recuerde que la dedicación es diaria y con ello el proceso de aprendizaje será placentero.

f) Debe constantemente esforzarse en controlar el tiempo, caso contrario el tiempo será un factor mortificante y dominante en su vida, generando estrés, ansiedad, insatisfacción, y en algunos casos somatizando y afectando su salud, además un bajo nivel de calidad en el trabajo académico.

g) No deje para el último sus actividades académicas, un trabajo realizado al apuro y sin el

tiempo suficiente, generalmente tiene resultado no deseado y desde luego consecuencias nada agradables.

Breve descripción de las características generales del documento a ser entregado (físico o virtual) que evidencia los resultados del trabajo académico.

Tamaño de hoja: formato A4 (21 $\times$ 29,7 centímetros).

Márgenes: todos los cuatro lados de la hoja tendrán 2,54 centímetros y todo el documento debe estar justificado con respecto a los márgenes.

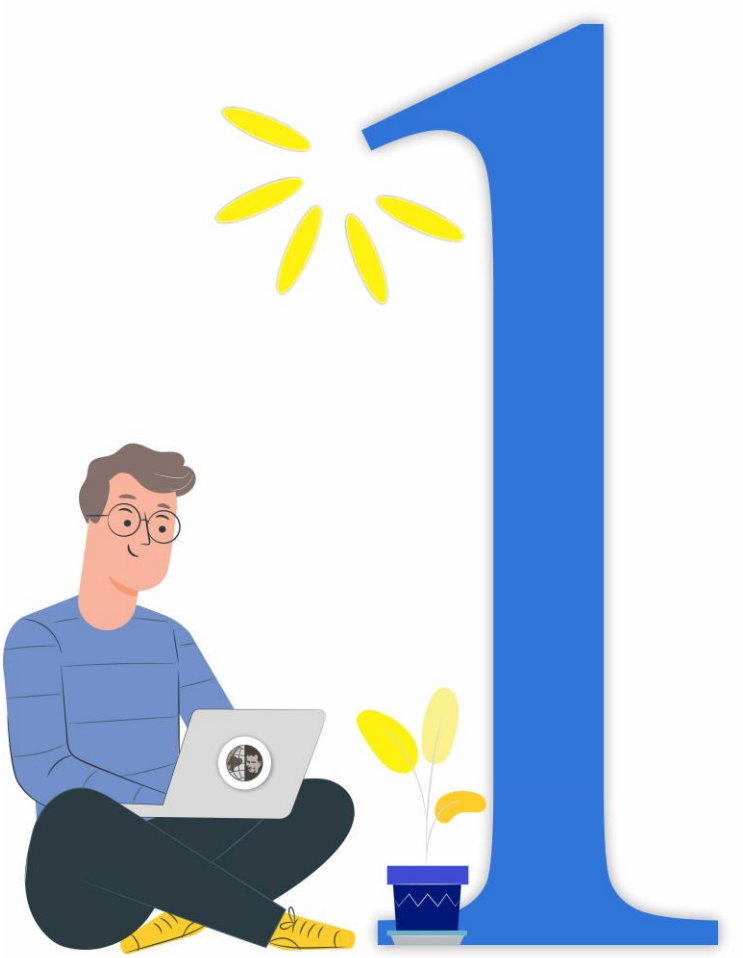
Espacio entre líneas y párrafos: Doble espacio y la alineación de los párrafos deberá ser justificado.

Tipo y tamaño de letra: Times New Roman, tamaño 12.

Sangría: Al inicio de cada párrafo, sangría de 5 espacios.

Numeración: inicia en la primera hoja y su ubicación es a la derecha parte superior.

¡Le auguro un buen trabajo y sobre todo el gusto de aprender, base para alcanzar el éxito y resultados aspirados!



Unidad 1

% Unidad 1

Capítulo 1 Desarrollo de la Unidad 1

"*Que el contenido de esta unidad abra las puertas del análisis, diseño y desarrollo para tu primer programa*". Tus profesores.

Unidad 1: Lógica Matemática

a) **Tiempo para su desarrollo:**

Total semanas:	3	**Total de horas:**	12
Fecha de inicio:		**Fecha finalización:**	
Componente Docencia	Componente Práctica de Aplicación y Experimentación	Componente Trabajo Autónomo	Total
12	6	12	30

b) **Presentación de la unidad**

Breve descripción de la unidad y los resultados de aprendizaje esperados.

c) **Actividades de aprendizaje estudiantil**

- Lectura, estudio, análisis de los contenidos relacionados a la unidad (bibliografía adjunta).
- Aplicación concreta de los contenidos conceptuales trabajados a través de la lectura.

d) **Recursos para las actividades de aprendizaje estudiantil**

- Texto 1 que consta al final de esta guía y/o archivo digital.
- Aplicación concreta de los contenidos conceptuales trabajados a través de la lectura.

e) **Cronograma sugerido**

Actividad académica	**Fecha de inicio**	**Fecha de finalización**	
Lectura de la temática			4 horas
Generación del trabajo académico a entregarse (Portafolio Estudiantil)			8 horas
Fecha de entrega al docente			

f) **Criterios para la evaluación de los trabajos académicos Unidad 1**

- Autoevaluación: Cada estudiante valora su actividad académica, de acuerdo con el formato establecido, tributa a ser justo y honesto con uno mismo.
- Evaluación: El docente valora el trabajo académico de cada estudiante, de acuerdo con el formato específico elaborado para el efecto. Tributa a dar ejemplo concreto a sus estudiantes, de los valores que debe practicar todo buen docente.
- Evaluación integral: Contribuye a una visión más cercana y equilibrada de la realidad evaluativa, contribuye a una evaluación formativa, que no busca sancionar sino, ante todo mejorar la realidad estudiantil involucrando valores y exigencia académica y de

esta manera contribuir a la Misión universitaria: formar honrados ciudadanos y buenos cristianos, con capacidad académica e investigativa que contribuyan al desarrollo sostenible local y nacional.

- De acuerdo con el artículo 41 del Reglamento de Régimen Académico de la Universidad Politécnica Salesiana, la evaluación tiene la característica de ser sistemática y continua, misma que considera un total de cien (100) puntos.

g) **Rúbrica de evaluación**

CRITERIO	DESCRIPCIÓN DEL CRITERIO	Excelente 80 %	Muy bueno 70 %	Bueno 60 %	Regular 40 %	Por mejorar menos de 40 %
Responsabilidad Académica	Experiencia concreta [Sentir] – Actuar					
	Reflexión [Analizar] - Observar					
Las actividades académicas las he realizado con:						
Responsabilidad formativa	Honestidad académica: sin plagio o copia	10 %	20 %	20 %	20 %	20 %
	Dedicación responsable	5 %	5 %	5 %	5 %	5 %
	Puntualidad en la entrega	5 %	5 %	5 %	5 %	5 %
TOTAL sobre 100 % del puntaje asignado						

Observaciones y retroalimentación	
Firma y nombre de:	
Estudiante	Profesor

1.1 Conceptualización [Profundizar] - Pensar

"*Que el contenido de esta unidad abra las puertas del análisis, diseño y desarrollo para tu primer programa*". Tus profesores.

1.1.1 ÁLGEBRA DE BOOLE

El álgebra de Boole se compone de un conjunto de axiomas algebraicos que mediante el uso de operaciones derivadas de leyes permiten simplificar expresiones equivalentes [**1**, **2**, **3**]. Comprender una expresión algebraica utilizando variables y operadores booleanos formando tablas de verdad permiten expresar problemas susceptibles a una formalización con un lenguaje [**4**, **5**, **6**, **7**]. De la idea de George Boole (1815-1864) de tratar operaciones lógicas y llevarlo a un sistema algebraico, logra componer variables abstractas utilizando operaciones definidas por las operaciones aritméticas, cumpliendo con propiedades iniciando una lógica formal matemática [**8**, **9**, **10**, **11**, **12**, **13**].

Boole logra representar los razonamientos (objeto de la lógica general) en un lenguaje formal con su semántica y validación rigurosa. Es por esto, que la lógica booleana le interesa el valor de la expresión evaluada como 0 o 1 en vez del significado concreto de dicha expresión. Si una expresión B se evalúa como 0, se escribirá $B = 0$.

El uso de operandos que representa la expresión simplificada en una variable corresponde a una letra de alfabeto en mayúscula. El operador booleano tiene una representación, ver tabla 1, que constituye a la compuerta lógica combinatoria, elemento fundamental de los circuitos electrónicos.

Tabla 1: *Operadores del Álgebra de Boole.*

Nombre de Operador	Símbolo	Lógica de enunciados	Lógica computacional
Negación	˜	$\neg$	Not
Suma o disyunción	+	$\vee$	Or
Producto o conjunción	·	$\wedge$	And
Conjunción opuesta		$\uparrow$	NAnd
Disyunción opuesta		$\downarrow$	NOr
Condicional	$\supset$	$\rightarrow$	
Bicondicional	$\equiv$	$\leftrightarrow$	XNOr
Disyunción exclusiva	$\oplus$	$\oplus$	XOr

Las tres primeras son los operadores simples (Or, And y Not), los siguientes se forman de los tres anteriores. En la tabla 2 podemos observar las tablas y sus simbologías. A partir de estos operadores se forman expresiones compuestas que pueden ser validadas por medio de una tabla de verdad. A continuación, en la tabla 3 se muestra un ejemplo.

Se llama álgebra de Boole a la expresión matemática formalizada por un conjunto A que tiene dos elementos 0 y 1, y se definen sobre ellos tres operaciones, la negación, la suma y el producto, que se rigen según las propiedades conmutativa, distributiva, identidad y complemento, siguiendo propiedades o axiomas. En la tabla cuatro observamos las propiedades.

Tabla 2: *Tablas de Operaciones Booleanas simples.*

A	$\sim$ A
	$\neg$ A
0	1
1	0

A	B	A · B
		A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A + B
		A $\vee$ B
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 3: *Operaciones Booleanas compuestas.*

		Or	And	NAnd	NOr	Implicación	XNOr	XOr
A	B	$\vee$	$\wedge$	$\uparrow$	$\downarrow$	$\rightarrow$	$\leftrightarrow$	$\oplus$
0	0	0	0	1	0	1	1	0
0	1	1	0	1	0	1	0	1
1	0	1	0	1	0	0	0	1
1	1	1	1	0	1	1	1	0

Tabla 4: *Propiedades del Álgebra de Boole.*

Conmutativa	A + B = B + A	A · B = B · A	
Asociativa	A + (B + C) = (A + B) + C	A · (B · C) = (A · B) · C	
Distributiva	A · (B + C) = (A · B) + (A · C)	A + (B · C) = (A + B) · (A + C)	
Identidad	A + 0 = A	A · 1 = A	
Dominancia	A + 1 = 1	A · 0 = 0	
Idempotencia	A + A = A	A · A = A	
Complemento	A + ($\sim$A) = 1	A · ($\sim$A) = 0	
Doble negación	$\sim \sim$A = A		
Absorción	A + (A · B) = A	A · (A + B) = A	A + ($\sim$A · B) = A + B
Leyes de Morgan	$\sim$ (A + B) = ($\sim$A) · ($\sim$B)	$\sim$ (A · B) = ($\sim$A) + ($\sim$B)	

Un ejemplo, obtenga las tablas de verdad de la siguiente expresión booleana: $(\sim B) \cdot (A + B)$ que puede expresarse así: $(\neg B) \wedge (A \vee B)$. Al utilizar las tablas de verdad desarrollamos los siguientes pasos (ver Tabla 5):

Tabla 5: *Construcción de la tabla de verdad para una expresión de ejemplo.*

		Paso 1	Paso 2	Paso 3
A	B	$\sim$B	(A+B)	($\sim$B) · (A+B)
		$\neg$B	(A $\vee$ B)	($\neg$B) $\wedge$ (A $\vee$ B)
0	0	1	0	0
0	1	0	1	0
1	0	1	1	1
1	1	0	1	0

Si utilizamos una calculadora lógica (Logic Calculator[1]) nos dará el mismo resultado (ver Fig. 1).

Por ejemplo, si tenemos las entradas de señales x,y,z, y un conjunto de salidas en la función f, como se muestra en la tabla 6.

[1]Disponible en https://sourceforge.net/projects/logiccalculator/?source=typ_redirect

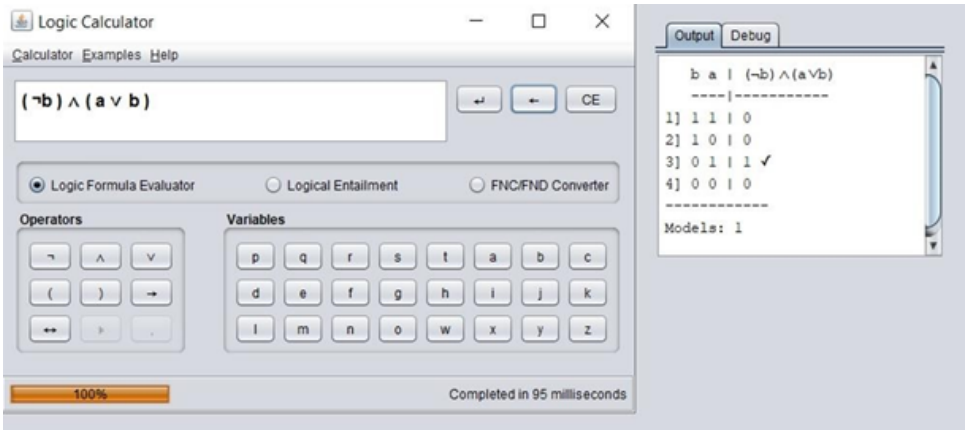


Figura 1: *Uso del programa Logic Calculator.*

Tabla 6: *Salidas de f en función de x,y,z.*

x	*y*	*z*	*f(x,y,z)*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

La salida de $f(x, y, z)$, puede representarse de la siguiente forma: $f(x, y, z) = \bar{x}\bar{y}z + \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xyz$, donde las representaciones con negado llevan una barra en la parte superior de la letra y son las simbologías utilizadas que representar $f(x, y, z) = 001 + 010 + 011 + 101 + 111$.

$$
\begin{aligned}
f(x,y,z) &= \bar{x}\bar{y}z + \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xyz && \text{Agrupando factores y reescribiendo la fórmula}\\
&= \bar{x}\bar{y}z + \bar{x}y(\bar{z} + z) + xz(\bar{y} + y) && \text{Factor común y complemento de la suma}\\
&= \bar{x}\bar{y}z + \bar{x}y(1) + xz(1) && \text{Aplicando identidad}\\
&= \bar{x}\bar{y}z + \bar{x}y + xz && \text{Aplicando absorción}\\
&= \bar{x}(y + z) + xz && \text{Aplicando distributiva}\\
&= \bar{x}z + \bar{x}y + xz && \text{Agrupando}\\
&= z(\bar{x} + x) + \bar{x}y && \text{Complemento de la suma}
\end{aligned}
$$

La función resultante sería $f(x, y, z) = z + \bar{x}y$. A este procedimiento se lo conoce como la aplicación de las reglas del álgebra de Boole.

Otro ejemplo, si nos piden verificar $xz + xyz = xz$ si es correcta la igualdad, podemos realizar aquellos pasos que nos permiten las reglas del álgebra de Boole. Por ejemplo, $xz(1 + y) = xz$, podemos sacar factor común y luego aplicar la propiedad de dominancia, dándonos $xz(1) = xz$, y por la propiedad de identidad obtener $xz = xz$, resultando el valor de la igualdad.

En la siguiente expresión en cambio, se deben utilizar otras reglas, como por ejemplo las Leyes de Morgan, $E = \overline{\left(\left(\bar{a}\bar{b}\right)c\right)}\ \overline{\left((\bar{a}+c)\left(\bar{b}+\bar{c}\right)\right)}$, entonces tenemos:

$$
\begin{aligned}
E &= \overline{\left(\left(\bar{a}\bar{b}\right)c\right)}\ \overline{\left((\bar{a}+c)\left(\bar{b}+\bar{c}\right)\right)} && \text{aplicamos la Leyes de Morgan y la involución.}\\
&= \left(\overline{\left(\left(\bar{a}\bar{b}\right)\right)}+\bar{c}\right)\ \left(\overline{((\bar{a}+c))}+\overline{\left(\left(\bar{b}+\bar{c}\right)\right)}\right) && \text{para eliminar paréntesis.}\\
&= (a+b+\bar{c})\,(a\bar{c}+bc) && \text{para eliminar paréntesis.}\\
&= aa\bar{c}+ba\bar{c}+\bar{c}a\bar{c}+abc+bbc+\bar{c}bc && \text{luego la ley distributiva.}\\
&= a\bar{c}+ba\bar{c}+\bar{c}a+abc+bc+0 && \text{transformamos las sumas de productos.}\\
&= a\bar{c}(1+b)+\bar{c}a+bc(a+1) && \text{agrupamos términos semejantes en parejas.}\\
&= a\bar{c}(1)+\bar{c}a+bc(1) && \text{aplicamos las reglas de dominancia.}\\
&= a\bar{c}+\bar{c}a+bc && \text{y luego de identidad y obtenemos la respuesta:}
\end{aligned}
$$

$$E = a\bar{c}+bc$$

Utilice la tabla 4 para determinar qué pasos debemos seguir cuando se debe reducir la expresión aplicando las reglas del álgebra de Boole.

1.1.2 TEORÍA DE PROPOSICIONES

El cálculo proposicional es una representación simbólica de la lógica clásica que sigue los principios de la **bivalencia**, la **no contradicción**, el **tercero excluido** y de **identidad**. La lógica requiere del cálculo proposicional ya que estudia las condiciones que cumple todo razonamiento para ser formalmente válido [**14**, **15**, **16**, **17**].

Un razonamiento es un conjunto de afirmaciones (dos o más) que producen otra en base a una inferencia, deducción o derivación de las primeras. Este razonamiento es válido si luego de un proceso estructural, lógico y correcto se evidencia una conexión que concluye en una afirmación resultado de las premisas [**18**, **19**, **20**, **21**].

Se debe diferenciar de lo que es verdad y de lo que tiene validez, por ejemplo:

Premisa 1: Los caballos (A) son reptiles (B)
Premisa 2: Los osos (C) son caballos (A)
Conclusión: Los osos (C) son reptiles (B)

Este razonamiento es válido formalmente, aunque sus premisas y su conclusión sean falsas. La lógica proposicional o de enunciados estudia la validez formal de los razonamientos. Una proposición es entendida en bloque cuando no se toma en cuenta los elementos que la integran, entendiendo esta como una unidad lingüística básica.
Del ejemplo anterior, "*los caballos son reptiles*" puede ser simbolizada de varios modos mediante el uso de la lógica, como:

Lógica proposicional: p = "*los caballos son reptiles*", p [se lee «p»] (es la proposición en bloque sin analizarla)

Lógica silogística: S = "*los caballos*" y P = "*son reptiles*", S -A- P [se lee «Todos los S son P] (analiza la proposición y diferencia el sujeto del predicado, analiza si el predicado se refiere a todos, algunos o a ninguno.

Lógica de predicados: $(Sx \rightarrow Px)$ [se lee «Para todo 'x' si 'x' es a 'S', entonces es a 'P'»] (analiza el tipo de relación que se da entre las propiedades atribuidas al sujeto de la proposición)

Una proposición al formar una estructura de un lenguaje natural, se la puede evaluar como verdadera o falsa. Ejemplos como: "*Todos los hombres son mortales*", "*El agua hierve a* $100°C$", o "*4 + 3 = 7*".
Para una mejor lectura de las proposiciones, se usan las letras minúsculas (preferible las finales del alfabeto) y empleando un subíndice podemos referirnos a ellas en mayores cantidades.

Por ejemplo, dadas las siguientes proposiciones:

p = La capital de la provincia del Guayas es Guayaquil
q = El número tiene una secuencia finita de decimales
r = El producto de dos números primos es un no primo

Se obtendría los valores de verdad:

v(p) = Verdadero (*dado que Guayaquil sí es la capital de la provincia del Guayas*)
v(q) = Falso (*la cantidad de decimales de es infinita*)
v(r) = Verdadero (*al multiplicar dos números primos se obtiene un no primo*)

Las proposiciones se clasifican en dos tipos, *simples* y *compuestas*. Las simples no contienen afirmaciones que las compongan, son conocidas como atómicas o variables proposicionales. A su vez se clasifican en, **simples abiertas** y **simples cerradas**.
Un ejemplo de una proposición simple abierta es: "A es una dama". Un ejemplo de una proposición simple cerrada es: "Cristina es una dama".

Las compuestas se construyen de las proposiciones simples, y de crear afirmaciones más complejas. Se conocen como moleculares. A su vez se clasifican en, **compuestas abiertas** y **compuestas cerradas**. Un ejemplo de una proposición compuesta abierta es: "Las A de Terranostra son blancas y las B de Arcadia son cremas". Un ejemplo de una proposición compuesta cerrada es: "Las casas de Terranostra son blancas y las villas de Arcadia son cremas".

Las tablas de verdad determinan los posibles resultados de las combinaciones en una estructura proposicional mediante un verdadero o falso. Sea L un lenguaje con dos proposiciones, p y q, al considerar el principio de bivalencia, p se puede representar mediante los valores de verdad de la siguiente manera (ver Tabla 7):

Tabla 7: *Tablas de valores de verdad para proposiciones.*

Bivalencia	**Negación**
p	$\neg$p
	$\sim$ p
F	V
V	F

Conjunción		
p	q	p $\wedge$ q
		p · q
F	F	F
F	V	F
V	F	F
V	V	V

Disyunción		
p	q	p $\vee$ q
		p + q
F	F	F
F	V	V
V	F	V
V	V	V

Implicación			
p	q	p $\rightarrow$ q	q $\rightarrow$ p
F	F	V	V
F	V	V	F
V	F	F	V
V	V	V	V

Bi implicación			
p	q	p $\leftrightarrow$ q	q $\leftrightarrow$ p
F	F	V	V
F	V	F	F
V	F	F	F
V	V	V	V

A través de las tablas de verdad, es factible establecer un procedimiento de prueba semántica que permita indicar si una expresión determinada es **tauotología, contradicción o contingencia**. Estas se representan en varias premisas y una conclusión, su fórmula es $(p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge p_5 \wedge p_6 \wedge \ldots \wedge p_n) \rightarrow C$. Si la conclusión en todas sus combinaciones de la fórmula es verdadera decimos que es tautología. Si la conclusión en todas sus combinaciones de la fórmula es falsa decimos que es una contradicción. Si la conclusión en todas sus combinaciones no es totalmente falsa ni verdadera decimos que es una contingencia.

Gracias al Álgebra de Boole y a la teoría de Proposiciones, podemos aplicar los teoremas a la lógica combinatoria mediante el uso de las compuertas lógicas (ver Tabla 8).

Tabla 8: *Propiedades del Cálculo Proposicional. V en mayúscula es verdadero*

Idempotencia de la conjunción y la disyunción	$(p \wedge p) \equiv p$	$(p \vee p) \equiv p$
Conmutativa de la conjunción y la disyunción	$(p \wedge q) \equiv (q \wedge p)$	$(p \vee q) \equiv (q \vee p)$
Asociatividad de conjunción y la disyunción	$[(p \wedge q) \wedge r] \equiv [p \wedge (q \wedge r)]$	$[(p \vee q) \vee r] \equiv [p \vee (q \vee r)]$
Distributiva	$[p \wedge (q \vee r)] \equiv [(p \wedge q) \vee (p \wedge r)]$	$[p \vee (q \wedge r)] \equiv [(p \vee q) \wedge (p \vee r)]$
Leyes de Morgan	$\neg(p \vee q) \equiv (\neg p \wedge \neg q)$	$\neg(p \wedge q) \equiv (\neg p \vee \neg q)$
Leyes de dominación	$p \vee V \equiv V$	$p \wedge F \equiv F$
Leyes de identidad	$p \wedge V \equiv p$	$p \vee F \equiv p$
Doble negación	$\neg(\neg p) \equiv p$	
Implicación	$(p \rightarrow q) \equiv (\neg p \vee q)$	
Co Implicación	$(p \leftrightarrow q) \equiv (p \rightarrow q) \wedge (q \rightarrow p)$	
Medio excluido	$p \vee \neg p \equiv V$	
Contradicción	$p \wedge \neg p \equiv F$	
Conversa de la implicación $(p \rightarrow q)$	$q \rightarrow p$	
Inversa de la implicación $(p \rightarrow q)$	$\neg p \rightarrow \neg q$	
Contrapositiva de la implicación $(p \rightarrow q)$	$\neg q \rightarrow \neg p$	Es la inversa de la conversa
Modus ponendus ponens	$((p \rightarrow q) \wedge p) \rightarrow q$	
Modus tollendo tollens	$((p \rightarrow q) \wedge \neg q) \rightarrow \neg p$	

Por ejemplo, una forma gráfica usando compuertas lógicas para representar una expresión algebraica: (X + Y) + Z = R, (ver Fig. 2).

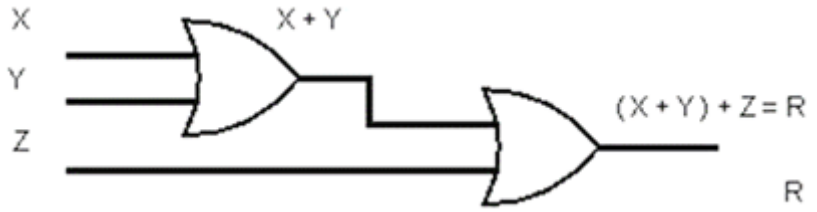


Figura 2: *Puertas lógicas OR combinadas.*

Otro uso de las compuertas conocidas como AND, OR, NOT y aquellas que se derivan de las anteriores, tenemos: $(X \cdot Y) \cdot Z = R$, (ver Fig. 3).

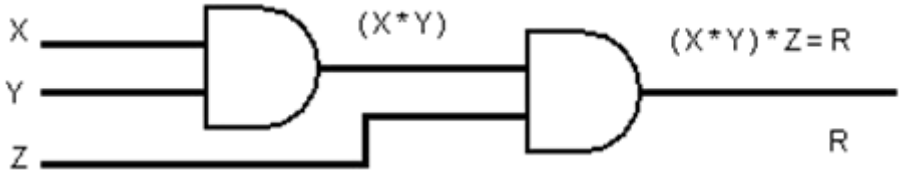


Figura 3: *Puertas lógicas AND combinadas.*

Determine la tabla de verdad a partir del siguiente circuito combinatorio, (ver Fig. 4).

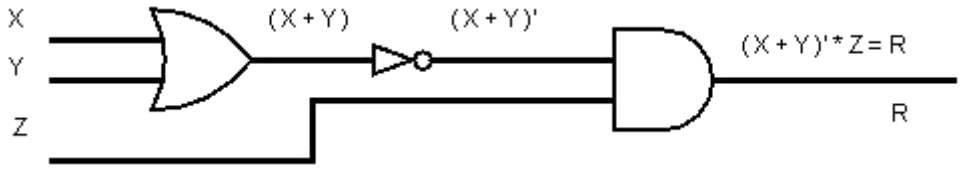


Figura 4: *Puertas lógicas combinadas.*

Se notan tres variables de entrada, X,Y y Z, una compuerta lógica Or que une X y Y, su resultado es negado y se une con Z en una compuerta And, dando como resultado R. La tabla de verdad sería (ver Tabla 9):

Tabla 9: *Resultado final de la tabla de verdad del circuito combinatorio.*

X	Y	Z	$(X+Y)$	$(X+Y)'$	$(X+Y)' * Z$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

Podemos determinar que las salidas de cada una de las combinaciones se obtiene un valor de verdad (aplicando las tablas en relación con la compuerta) y la suma de todas en conjunto (valor final de R) no son todas Falsas (contradicción), existe al menos una que es distinta dando como resultado una contingencia. Si todas las respuestas hubieran sido Verdaderas se obtendría una tautología. Para generar las gráficas puedes usar un software gratuito llamado Logisim[2].

[2]Disponible Logisim desde `http://www.cburch.com/logisim/index_es.html`

1.2 Experiencia activa [Experimentar] - Hacer

"*Que el contenido de esta unidad abra las puertas del análisis, diseño y desarrollo para tu primer programa*". Tus profesores.

Sean las siguientes proposiciones:

p = "Alemania está en Europa"
q = "China está en Asia"

Escribe las proposiciones según el contenido de p y q:

1. $\sim q$
2. $p \Rightarrow q$
3. $\sim (p \vee q)$
4. $\sim (p \wedge q)$

Tabla 10: *Representación de las proposiciones y su conclusión.*

	Entrada 1	Entrada 2	Proposición	Conclusión
1	q		$\sim$ q	"Alemania NO está en Asia"
2	p	q	p $\Rightarrow$ q	"Alemania está en Europa" entonces "China está en Asia"
3	p	q	$\sim$ (p $\vee$ q) $\equiv$ ($\sim$ p $\wedge \sim$ q)	"Alemania NO está en Europa" y China NO está en Asia"
$\wedge$	p	q	$\sim$ (p $\wedge$ q) $\equiv$ ($\sim$ p $\vee \sim$ q)	"Alemania NO está en Europa" o China NO está en Asia"

Sean las siguientes proposiciones:

a = "El colibrí es un ave"
b = "A Juan le gusta escuchar a los Beatles"

Escribe en forma simbólica los siguientes enunciados:

1. El colibrí es un ave y a Juan le gusta escuchar a los Beatles
2. El colibrí es un ave y a Juan no le gusta escuchar a los Beatles
3. El colibrí no es un ave y a Juan le gusta escuchar a los Beatles
4. No es verdad que el colibrí es un ave y que a Juan le gusta escuchar a los Beatles

Tabla 11: *Representación de las proposiciones y su simbología.*

	p	q	Simbología
1	El colibrí es un ave	a Juan le gusta escuchar a los Beatles	p $\wedge$ q
2	El colibrí es un ave	a Juan le no gusta escuchar a los Beatles	p $\wedge \sim$q $\equiv \sim$($\sim$p $\vee$ q)
3	El colibrí no es un ave	a Juan le gusta escuchar a los Beatles	$\sim$p $\wedge$ q
4	No es verdad que el colibrí es un ave	a Juan le gusta escuchar a los Beatles	$\sim$(p $\vee \sim$q) $\equiv \sim$p $\wedge$ q

Determina la conversa, contrapositiva e inversa de las siguientes implicaciones:

a) $p \Rightarrow q =$ "Si 2 es divisor de 4, entonces no es par"
b) $p \Rightarrow q =$ "Si x es múltiplo de 6, entonces es divisor de 36"

Tabla 12: *Representación de la Inversa, Conversa y Contrapositiva.*

Variable 1	Variable 2	EXPRESION	INVERSA	CONVERSA	CONTRAPOSITIVA
p	q	$p \Rightarrow q$	$\sim p \Rightarrow \sim q$	$q \Rightarrow p$	$\sim q \Rightarrow \sim p$
2 es divisor de 4	no es par"	Si 2 es divisor de 4, entonces no es par	Si 2 no es divisor de 4, entonces 2 no es par	No es par entonces 2 es divisor de 4	Si 2 es par entonces 2 no es divisor de 4

Variable 1	Variable 2	EXPRESION	INVERSA	CONVERSA	CONTRAPOSITIVA
p	q	$p \Rightarrow q$	$\sim p \Rightarrow \sim q$	$q \Rightarrow p$	$\sim q \Rightarrow \sim p$
x es múltiplo de 6	es divisor de 36	Si x es múltiplo de 6, entonces es divisor de 36	Si x no es múltiplo de 6, entonces no es divisor de 36	Es divisor de 36, entonces x es múltiplo de 6	No es divisor de 36, entonces x no es múltiplo de 6

Construye la tabla de verdad para cada una de las siguientes proposiciones y determina si es una Tautología, Contradicción o Contingencia:

$(p \vee q) \wedge \sim (p \Rightarrow q)$

Tabla 13: *Resultado de la tabla de verdad del ejercicio.*

p	q	$(p \vee q)$	$(p \Rightarrow q)$	$\sim (p \Rightarrow q)$	$(p \vee q) \wedge \sim (p \Rightarrow q)$
0	0	0	1	0	0
0	1	1	1	0	0
1	0	1	0	1	1
1	1	1	1	0	0

Es contingencia.

$(p \Rightarrow q) \vee (q \Rightarrow p)$

Tabla 14: *Resultado de la tabla de verdad del ejercicio.*

p	q	$(p \Rightarrow q)$	$(q \Rightarrow p)$	$(p \Rightarrow q) \vee (q \Rightarrow p)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1

Es tautología.

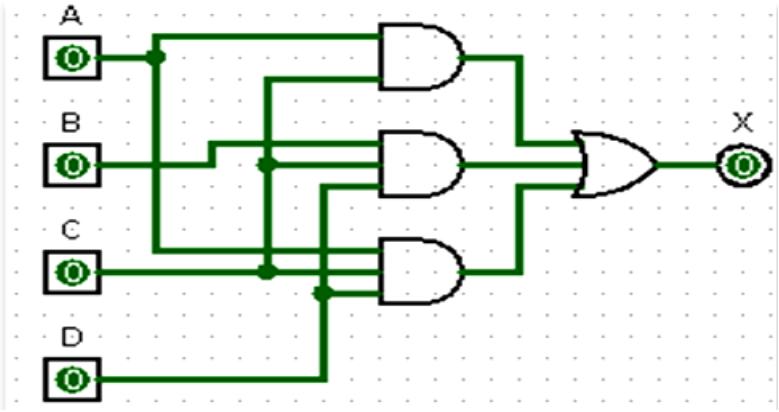
$(p \wedge (p \Rightarrow q)) \Rightarrow p$

Tabla 15: *Resultado de la tabla de verdad del ejercicio.*

p	q	$(p \Rightarrow q)$	$(p \wedge (p \Rightarrow q))$	$(p \wedge (p \Rightarrow q)) \Rightarrow p$
0	0	1	0	1
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

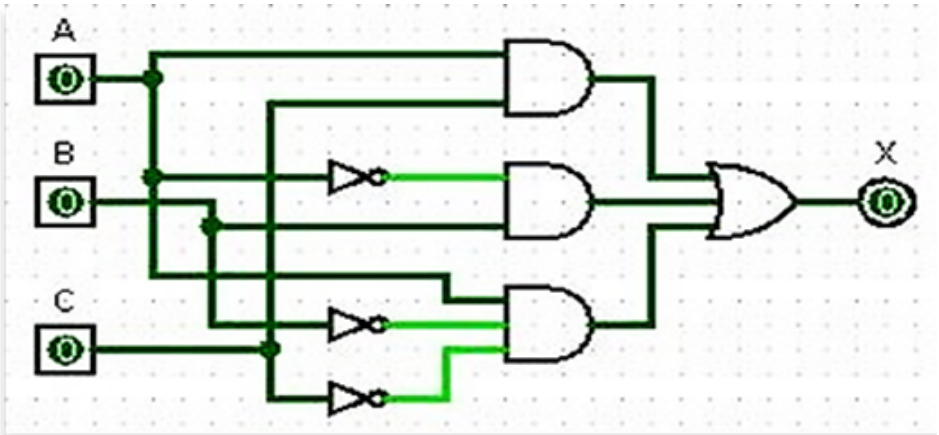
Es tautología.

Dada la siguiente expresión booleana **AC + BCD + ACD** grafique el circuito lógico resultante.



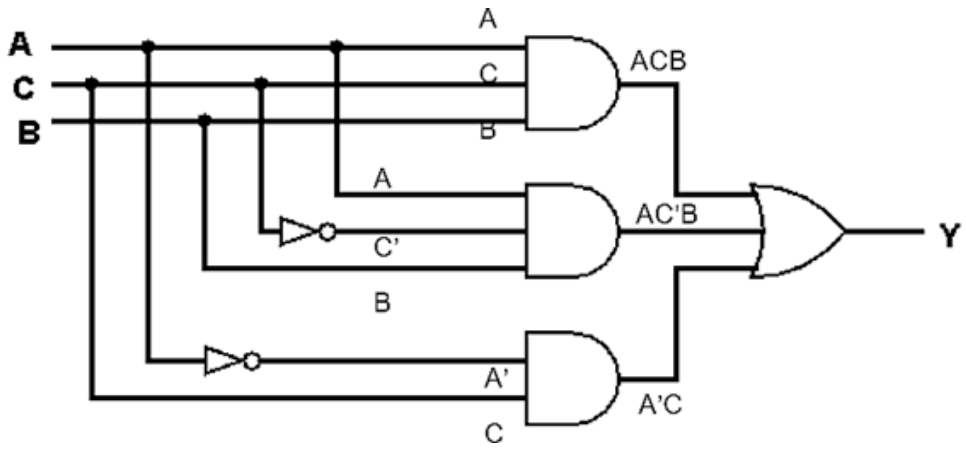
Dada la siguiente expresión booleana $X = AC + \bar{A}B + A\overline{BC}$ grafique el circuito lógico resultante

	A	B	C	AC	$\bar{A}$	$\bar{A}B$	BC	$\overline{BC}$	$A\overline{BC}$	X
1.	0	0	1	0	1	0	0	0	0	0
2.	0	0	0	0	1	0	0	0	0	0
3.	0	1	1	0	1	1	1	1	0	1
4.	0	1	0	0	1	1	0	1	0	1
5.	1	0	1	1	0	1	0	1	0	1
6.	1	0	0	0	0	1	0	1	0	1
7.	1	1	1	1	0	0	1	1	0	0
8.	1	1	0	0	0	0	0	1	0	1



Utilizando el siguiente circuito realice lo siguiente:

Escriba la expresión booleana para el siguiente circuito. Luego construya la tabla de verdad. Simplifique la expresión booleana de salida del circuito utilizando las leyes del algebra de Boole.



Entonces observamos las entradas a cada una de las compuertas, por ejemplo, la primera compuerta AND tiene como salida ACB, la segunda compuerta tiene de salida $AC'B$, y la tercera compuerta tiene como salida $A'C$, con ello realizamos la suma en la salida Y, gracias a la compuerta OR. Teniendo como resultado $Y = ACB + AC'B + A'C$. Es posible reducir la expresión teniendo $Y = AB(C + C') + A'C$, y luego $Y = AB(1) + A'C$. Obteniendo finalmente $Y = AB + A'C$. La tabla de verdad se presenta a continuación:

A	B	C	AB	A'	C	A'C	**Y**
0	0	0	0	1	0	0	0
0	0	1	0	1	1	1	1
0	1	0	0	1	0	0	0
0	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0
1	1	0	1	0	0	0	0
1	1	1	1	0	1	0	0

Teniendo como resultado una contingencia.

1.3 Experiencia concreta [Sentir] – Actuar

"*Que el contenido de esta unidad abra las puertas del análisis, diseño y desarrollo para tu primer programa*". Tus profesores.

1.3.1 DESAFÍO

Desafío 1. Representación Lógica de Enunciados

Representar con notación lógica cada uno de los siguientes enunciados:

Si tengo conocimientos de computación y domino el inglés, entonces no tendré problemas para encontrar trabajo. Si tengo problemas para encontrar trabajo, entonces tengo más de 40 años o no me preparé lo suficiente. Por lo tanto, si me preparo lo suficiente y no tengo más de 40 años y domino el inglés, entonces no tendré problemas para encontrar trabajo.

A: tengo conocimientos de computación
B: domino el inglés
C: tendré problemas para encontrar trabajo
D: tengo más de 40 años
E: *no me preparé lo suficiente*

$[(A \wedge B) \Rightarrow \neg C)] \wedge [(C \Rightarrow (D \vee E)] \Rightarrow [(\neg E \wedge \neg D \wedge B) \Rightarrow \neg C]$
¿Puedes escribir otra forma de representación lógica, observa E?

Desafío 2. Enunciado de una Representación Lógica

Representar cada una de las notaciones lógicas en forma de enunciados:

Notación lógica: $[((p \wedge q) \Rightarrow \neg r) \vee (r \Leftrightarrow s)] \Rightarrow [(r \wedge s) \Rightarrow (\neg p \vee \neg q)]$

Para el enunciado considerar las siguientes proposiciones: Compré un automóvil y me levanté más temprano entonces no llego tarde al trabajo, o llego tarde al trabajo sí y solo si fui despedido. Entonces si llego tarde al trabajo y fui despedido entonces no compré un automóvil o no me levanté más temprano.

Compré un automóvil **y** me levanté más temprano entonces no llego tarde al trabajo , **o** llego tarde al trabajo sí y solo si fui despedido . Entonces si llego tarde al trabajo **y** fui despedido entonces no compré un automóvil **o** no me levanté más temprano.

p: Compré un automóvil
q: Me levanté más temprano
r: Llego tarde al trabajo

s: Fui despedido

¿Puedes reescribir otro enunciado que se asemeje a una historia que tú puedas aplicar la simbología dada?

Desafío 3. Aplicación de las reglas del Álgebra de Boole y su valor de verdad

Con el siguiente enunciado, construye la tabla de verdad, determina si es contingencia, tautología o contradicción, finalmente construye el circuito correspondiente.

$$[(p \wedge \sim q) \vee \sim (q \wedge \sim p)]$$

1.4 Reflexión [Analizar] - Observar

"*Que el contenido de esta unidad abra las puertas del análisis, diseño y desarrollo para tu primer programa*". Tus profesores.

1.4.1 EVALUACIÓN

[Lea la página 4 y siga el formato para la entrega de todas las actividades, en un solo documento]

Validación del aprendizaje de la Unidad 1

- Elabora tres enunciados que represente una historia en base al contexto que vives. Luego representa en una simbología lógica. Determina las proposiciones que sean necesarias agregar con letras mayúsculas desde la A, B, etc. Tu enunciado debe contener al menos dos premisas y una conclusión.
- Elabora luego el conjunto de puertas combinatorias que se obtendría de su expresión. Finalmente elabora la tabla de verdad del conjunto utilizadas para determinar si es contingencia, contradicción o tautología.

Para este trabajo es necesario que utilices los programas Logic calculator[3] y Logisim[4].

Reactivos de la Unidad 1

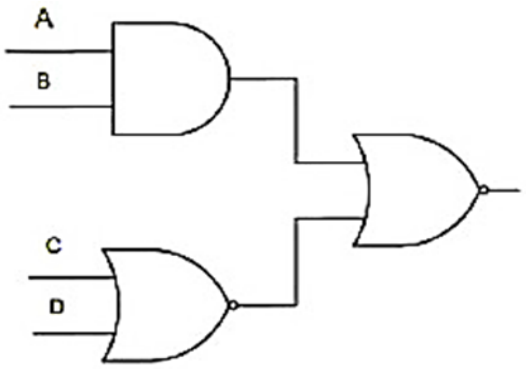
Pregunta 1. Represente con notación lógica el siguiente enunciado y determine su resultado de verdad (de su expresión encontrada) si es verdadera o falsa, cuando todas las entradas son falsas: Si se realiza un buen diseño de la base de datos y se hace una buena programación, entonces se accederá rápidamente la información. Si no se hace buena programación, entonces toma mucho tiempo corregir el programa. Por lo tanto, si no se accede rápidamente la información y toma mucho tiempo corregir el programa, entonces no se ha realizado un buen diseño de la base de datos. Escoja una de las opciones.

$[(p \wedge q \leftrightarrow r) \wedge (\sim p \Rightarrow s)] \Rightarrow [(\sim r \wedge s) \Rightarrow \sim p]$] resultado V cuando p=q=r=s= F	
$[(\sim p \vee q \Rightarrow r) \wedge (\sim p \vee s)] \Rightarrow [(\sim r \vee s) \Rightarrow p]$] resultado V cuando p=q=r=s= F	
$[(p \wedge q \Rightarrow r) \wedge (\sim p \Rightarrow s)] \Rightarrow [(\sim r \wedge s) \Rightarrow \sim p]$] resultado F cuando p=q=r=s= F	

Pregunta 2. Dado el siguiente circuito combinatorio, determine la salida. Escoja una de las opciones.

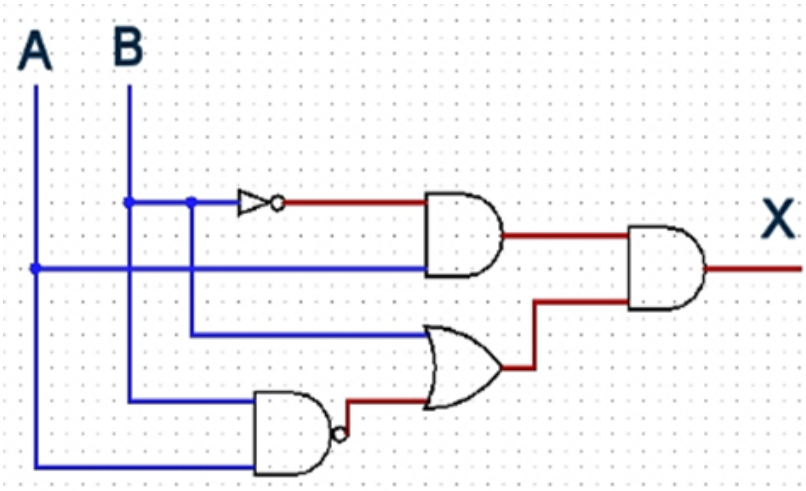
[3]Disponible en https://sourceforge.net/projects/logiccalculator/

[4]Disponible en http://www.cburch.com/logisim/



$no((A * B) + no(C + D))$	
$no(A + B) + (C + D)$	
$(A * B) * no(C * D)$	

Pregunta 3. Seleccione los valores de las entradas que debe tener A, B para que la salida X sea 1. Escoja una de las opciones.



A=0, B=0	
A=0, B=1	
A=1, B=1	

Pregunta 4. Simplificar la siguiente expresión con álgebra de Boole.

$$\left(A + C\bar{A}\right)\overline{(B + C)\,(C + A)}$$

$A \cdot \bar{A} \cdot \bar{C}$	
$(AC + B)$	
$\bar{A}(\bar{B} + \bar{C})$	

¡Buen trabajo!

Unidad 2

% Unidad 2

Capítulo 2 Desarrollo de la Unidad 2

"*Que el contenido de esta unidad sea un desafío de representar algoritmos mediante una adecuada solución*". Tus profesores.

Unidad 2: Algoritmos

a) **Tiempo para su desarrollo:**

Total semanas:	5	**Total de horas:**	20
Fecha de inicio:		**Fecha finalización:**	
Componente Docencia	Componente Práctica de Aplicación y Experimentación	Componente Trabajo Autónomo	Total
20	10	20	50

b) **Presentación de la unidad**

Breve descripción de la unidad y los resultados de aprendizaje esperados.

c) **Actividades de aprendizaje estudiantil**

- Lectura, estudio, análisis de los contenidos relacionados a la unidad (bibliografía adjunta).
- Aplicación concreta de los contenidos conceptuales trabajados a través de la lectura.

d) **Recursos para las actividades de aprendizaje estudiantil**

- Texto 1 que consta al final de esta guía y/o archivo digital.
- Aplicación concreta de los contenidos conceptuales trabajados a través de la lectura.

e) **Cronograma sugerido**

Actividad académica	**Fecha de inicio**	**Fecha de finalización**	
Lectura de la temática			6 horas
Generación del trabajo académico a entregarse (Portafolio Estudiantil)			14 horas
Fecha de entrega al docente			

f) **Criterios para la evaluación de los trabajos académicos Unidad 2**

- Autoevaluación: Cada estudiante valora su actividad académica, de acuerdo con el formato establecido, tributa a ser justo y honesto con uno mismo.
- Evaluación: El docente valora el trabajo académico de cada estudiante, de acuerdo con el formato específico elaborado para el efecto. Tributa a dar ejemplo concreto a sus estudiantes, de los valores que debe practicar todo buen docente.
- Evaluación integral: Contribuye a una visión más cercana y equilibrada de la realidad evaluativa, contribuye a una evaluación formativa, que no busca sancionar sino, ante todo mejorar la realidad estudiantil involucrando valores y exigencia académica y de

esta manera contribuir a la Misión universitaria: formar honrados ciudadanos y buenos cristianos, con capacidad académica e investigativa que contribuyan al desarrollo sostenible local y nacional.

- De acuerdo con el artículo 41 del Reglamento de Régimen Académico de la Universidad Politécnica Salesiana, la evaluación tiene la característica de ser sistemática y continua, misma que considera un total de cien (100) puntos.

g) **Rúbrica de evaluación**

CRITERIO	DESCRIPCIÓN DEL CRITERIO	Excelente 80 %	Muy bueno 70 %	Bueno 60 %	Regular 40 %	Por mejorar menos de 40 %
Responsabilidad Académica	Experiencia concreta [Sentir] – Actuar					
	Reflexión [Analizar] - Observar					
Las actividades académicas las he realizado con:						
Responsabilidad formativa	Honestidad académica: sin plagio o copia	10 %	20 %	20 %	20 %	20 %
	Dedicación responsable	5 %	5 %	5 %	5 %	5 %
	Puntualidad en la entrega	5 %	5 %	5 %	5 %	5 %
TOTAL sobre 100 % del puntaje asignado						

Observaciones y retroalimentación	
Firma y nombre de:	
Estudiante	Profesor

2.1 Conceptualización [Profundizar] - Pensar

"*Que el contenido de esta unidad sea un desafío de representar algoritmos mediante una adecuada solución*". Tus profesores.

2.1.1 ALGORITMOS

Un algoritmo es un conjunto de pasos ordenados conocidos como instrucciones que permiten realizar una tarea específica para obtener un resultado deseado, la solución de un problema o un fin determinado [**22**, **23**, **24**, **25**].

Para el diseño de un algoritmo se requieren operaciones básicas que utilizan datos de entrada para generar una salida. Estos datos son valores constantes y variables. Se relacionan por medio de operadores (lógicos «booleanos», aritméticos y relacionales). Los valores que se asignan a las variables pueden generar nombres especiales como acumuladores, contadores, incrementadores, decrementadores, inicializadores, banderas entre otros [**26**, **27**, **28**, **29**, **30**, **31**, **32**].

Cada paso de un algoritmo se lo puede conocer como una instrucción. Las instrucciones pueden ser sentencias de asignación, condicionadas o repetitivas [**33**, **34**, **35**, **36**, **37**]. Al combinar las operaciones básicas se obtienen estructuras conocidas como, estructura secuencial, estructura selectiva o estructurada repetitiva [**38**, **39**, **40**].

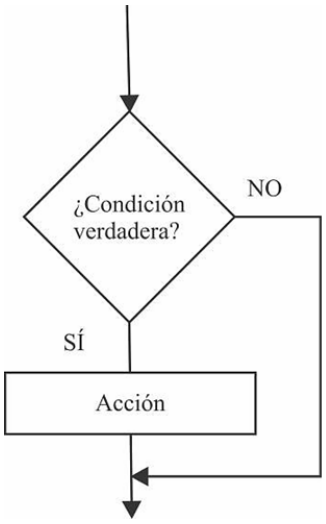


Figura 5: *Estructura Selectiva Simple.*

Las estructuras condicionales, nos permiten validar el valor de una variable por medio de una condición. Es la elección de uno u otro camino que depende de una respuesta. Se encuentran, la estructura simple, doble, anidada o múltiple.

La estructura selectiva simple o *sí simple* (ver Fig. 5), es aquella que del valor verdadero de la condición se ejecuta una o más acciones (instrucciones), si la condición no es verdadera (falso) no se ejecuta acción alguna.

Cuando existen acciones por el valor de falsedad (NO) de la condición, se conoce a la estructura selectiva como una estructura selectiva doble, *sí-sino* (ver Fig. 6).

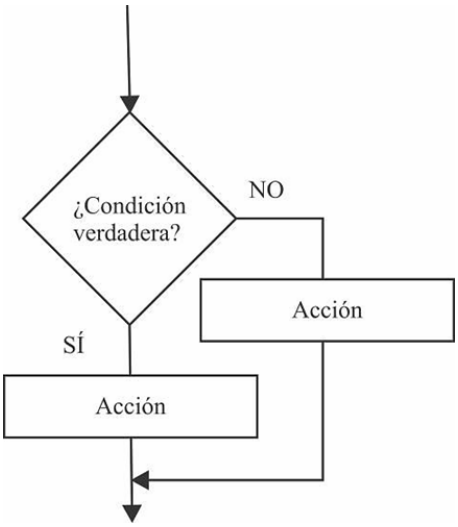


Figura 6: *Estructura Selectiva Doble.*

Cuando haya más condiciones por desarrollar, es decir, deben cumplir un conjunto de ellas (una tras otra), luego de que una a una vaya apareciendo y resolviendo (ver Fig. 7) se debe utilizar una estructura de anidada de condicionales. A esta estructura se la conoce como *sí anidado*.

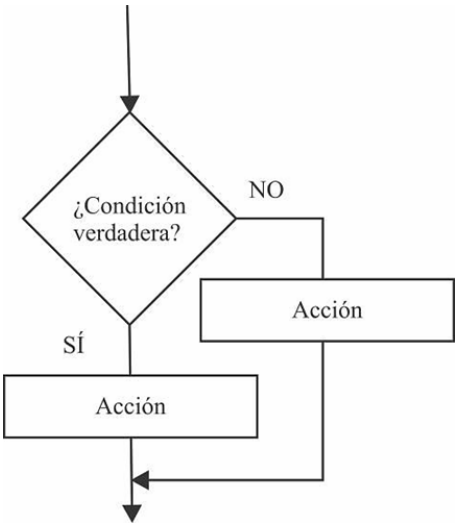


Figura 7: *Estructura Anidada.*

El flujo de la estructura anidada depende de cómo se vaya cumpliendo cada respuesta en la bifurcación, es decir si se observa la figura 7, si la primera condición es falsa, se realiza la segunda condición (segundo rombo hacia abajo). La utilidad de estas estructuras se las puede considerar al tratar condición por condición. Pueden ser sustituidas con el uso de operadores lógicos que se ve en el Álgebra de Boole y que serán de utilidad en las consultas que se realicen en las bases de datos al extraer información de ellas.

Las estructuras repetitivas, desarrollan o ejecutan, una o más acciones, tenemos tres estructuras repetitivas a estudiar. La estructura *mientras*, la estructura *repetir hasta* o *hacer mientras* y la

estructura *para*. Todas ellas, las tres, realizan lo mismo, iterar o repetir una línea o bloque de líneas. En problemas complejos por resolver, las estructuras repetitivas pueden ayudarnos a encontrar la solución, como por ejemplo cálculos recurrentes, y por medio de ellas la codificación se reduce a pocas líneas de código.

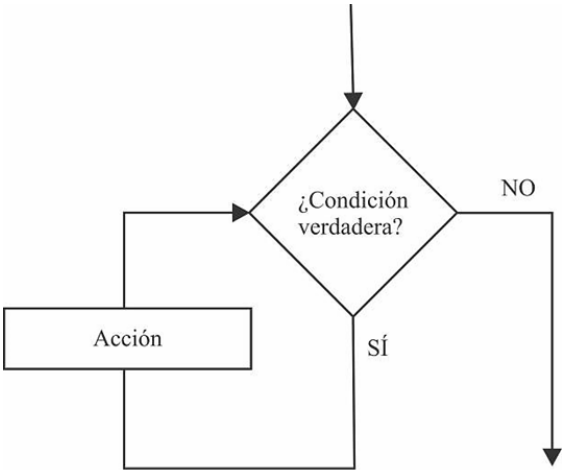


Figura 8: *Estructura Repetitiva Mientras.*

La estructura repetitiva, *mientras*, desarrolla o ejecuta, una o más acciones, si la condición es verdadera, sino lo es sale del lazo o bucle y finaliza la ejecución de la estructura o en el caso a estudiar, no ingresa a ella. Esta estructura se la conoce como una estructura restrictiva debido a que si al llegar a su condición, esta no se cumple, no desarrolla acción alguna y continúa (ver Fig. 8). Esta tiene dos variantes que se las conoce por el tipo de dato a validar (ver Fig. 9), es decir, por la condición a evaluar se conoce el tipo de estructura utilizada. Estas dos son, *la estructura mientras controlada por una variable contador y la estructura mientras controlada por una variable centinela*.

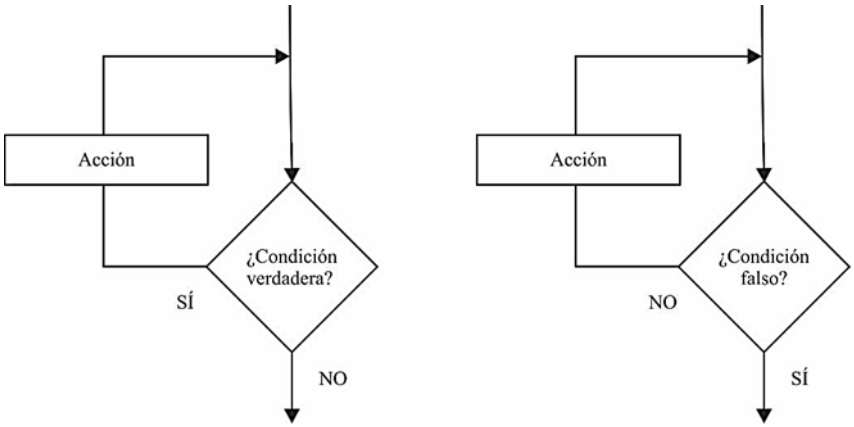


Figura 9: *Estructura Repetitiva Hacer Mientras o Repetir hasta.*

La estructura repetitiva, *hacer mientras* o *repetir hasta* es una estructura que su condición se encuentra al final de la estructura. Si la condición pregunta por verdadero se la conocerá como estructura hacer mientras, en cambio si la condición pregunta por falso se la conocerá como repetir hasta. Para entender esto desde un proceso algorítmico, la estructura *repetir hasta* fue muy utilizada hace años, cuando llegaron lenguajes de programación como Java, esta estructura se la cambió a *hacer mientras*, para repetir por verdadero como ya se lo realizada con la estructura *mientras*,

pero al aparecer nuevos lenguajes como Python, ésta ya no es utilizada, debido a que como las tres hacen lo mismo esta estructura no se muestra a simple vista

La utilidad de la estructura *hacer mientras* o *repetir hasta* es en la creación de los menús, ya que esta estructura presenta una acción o acciones por lo menos una vez y luego se pregunta por la condición. En *repetir hasta* si es falsa la condición repetirá las instrucciones en un primer caso. Si es verdadera finalizará, continuará o saldrá de la estructura. En *hacer mientras* si es verdadera la condición repetirá las instrucciones en un segundo caso. Si es falsa finalizará, continuará o saldrá de la estructura (ver Fig. 9).

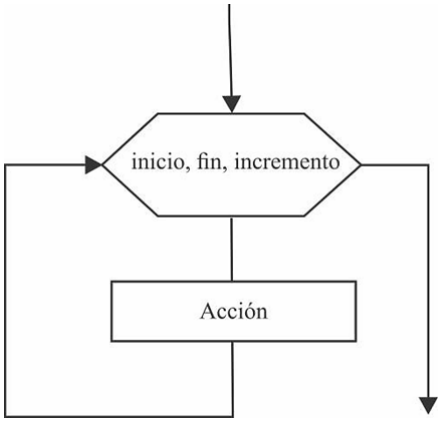


Figura 10: *Estructura Repetitiva Para.*

La estructura repetitiva *para*, tiene una particularidad que se la define como aquella estructura que se utiliza cuando se sabe la cantidad de repeticiones a realizar, es decir cuando se sabe el número exacto de repeticiones es mejor usar el lazo *para* (ver Fig. 10).

Para comprender el uso de las estructuras selectivas y de repetición se debe entonces enfocar en la condición, aquella expresión que utiliza operadores de relación, lógicos (booleanos) y aritméticos para determinar su resultado de verdad o falsedad.

El uso de los operadores, relacionales, lógicos y aritméticos forman parte de la resolución de los algoritmos con sus debidos ejemplos e interpretaciones que observamos en la Tabla 16.

Tabla 16: *Operadores de relación, lógicos y aritméticos.*

Operador relacional		Nombre		Ejemplo		Interpretación	
<		Menor		x <y		x es menor a y	
>		Mayor		x >y		x es mayor a y	
<=		Menor e igual		x <= y		x es menor e igual a y	
>=		Mayor e igual		x >= y		x es mayor e igual a y	
==		Es igual		x == y		x es igual a y	
!=		No es igual		x != y		x es no es igual a y	
Operador lógico							
&&	And	Y		(x <w) && (x >y)		x es menor a w y x es mayor a y	
||	Or	O		(x >w) || (x <y)		x es mayor a w o x es menor a y	
!	Not	no		!(x <w)		x no es menor a w	
Operador aritmético							
+	Suma	*	Multiplicación	+=	Suma un valor a una variable	//=	División entera de una variable
–	Resta	/	División	–=	Resta un valor a una variable	**=	Potencia un valor a una variable
//	División entera	%	Módulo o resto	*=	Multiplica un valor a una variable	%=	Resto de la división de una variable
**	Potencia			/=	Divide un valor a una variable		

De esta tabla, podemos observar, que los operadores relacionales son aquellos que nos permiten comparar entre un conjunto de valores o variables, entre ellos el menor, el mayor, menor igual, mayor igual, el igual (doble signo) y el signo de no es igual (en varios lenguajes puede representarse de formas distintas). El signo de no es igual, en PseInt es representado por $<>$ o por $\neq$. Para los lenguajes de programación se muestra con dos signos juntos $!=$ que equivale a no es igual.

Los operadores lógicos son conocidos como los operadores booleanos, estos operadores son las representaciones de las puertas lógicas que se revisa en la unidad 1, asimismo para los lenguajes de programación usan operadores como doble ampersand, &, o conjugación copulativa que representan a la compuerta And. En cambio, para la compuerta Or, se usa una doble Pleca o barra vertical. Finalmente, para el representar al operador de negación, se utiliza el signo de admiración cerrado. Cada uno de estos operadores deben ser usados en las líneas o sentencias correcto de acuerdo con una sintaxis, es decir deben estar en el lugar apropiado y de acuerdo con la lógica algorítmica o programática, ellos pueden dar un resultado de verdadero o falso, y ante esto el programador deber relacionar dicha salida o valor resultante de acuerdo con la línea escrita.

Los operadores aritméticos utilizados como en las calculadoras son los más comunes, pero cuando estamos en un lenguaje debemos asegurarnos de conocer cómo es su aplicabilidad. Por ejemplo, en algunos lenguajes, separan la división entera de la división con punto flotante o punto decimal. Es decir, usan la barra inclinada, de división o del operador cociente para obtener la división de dos números, pero al tratarse de la parte entera de una división, algunos lenguajes utilizan la doble barra inclinada ($/$). Con esto evitamos realizar otras operaciones como un redondeo hacia arriba o hacia abajo, solo se separa la parte entera. En PseInt se usa una función llamada TRUNC, de igual manera que en otros leguajes.

Para obtener el resto de una división o el módulo, se utiliza el signo del porcentaje. Con ello podemos realizar la operación de obtención del residuo. A medida que profundices con un lenguaje u otro, irás encontrando estructuras de corta referencia como el uso de dos operadores como $+=$ o $**$ donde ellos representan operaciones de cierta expresión aritmética pero ahora con una estructura acortada.

Por ejemplo, tener una sentencia como $i+=1$ es como tener $i = i + 1$ donde a la variable i se le incrementa 1 en la siguiente línea. Tener una variable $c = 3 * *2$ es como indicar que la variable c tiene asignado el valor de 3 al cuadrado. Todo esto de acuerdo con la sintaxis del lenguaje y las posibilidades que tiene para el desarrollador. Te dejamos la tabla 16 como una referencia a los posibles operadores que irás encontrando a lo largo del estudio de programación. En los siguientes ejercicios presta atención a los ejemplos, en ellos se irán incorporando líneas de sentencias donde encontrarás estas nomenclaturas.

2.2 Experiencia activa [Experimentar] - Hacer

"*Que el contenido de esta unidad sea un desafío de representar algoritmos mediante una adecuada solución*". Tus profesores.

Para los siguientes ejercicios, se usa el software PSeInt[5], con este programa elabora el algoritmo en Pseudocódigo y tiene una opción para presentar el Diagrama de Flujo.

Escriba un algoritmo en *PSeInt* y su representación en su diagrama de flujo para resolver el problema para calcular el área del círculo, ingresando el valor del radio por teclado, presente la respuesta en pantalla y utilice el valor del PI con 3.1415.

```
1 Algoritmo ejercicio01
2     Escribir 'Ingrese el radio: '
3     Leer ra
4     valor_PI <- 3.1415
5     area <- valor_PI*ra*ra
6     Escribir 'Respuesta: ',area
7 FinAlgoritmo
```

Figura 11: *Escritura del Seudocódigo en PSeInt.*

La imagen anterior (ver Fig. 11), el ejercicio01 muestra la escritura en seudocódigo la respuesta del problema. El mismo programa presenta la opción de representar dicho algoritmo en diagrama de flujo (ver Fig. 12).

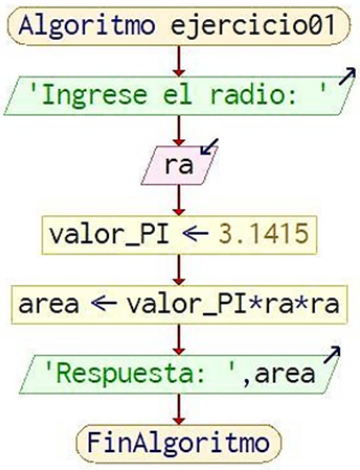


Figura 12: *Representación en diagrama de flujo del algoritmo en PSeInt.*

La respuesta que podemos observar del diagrama es el uso de figuras geométricas, el paralelogramo para representar la entrada (color rosa) y salida (color verde) de datos. Los rectángulos (color amarillo) permiten la ejecución de procesos asignando valores constantes o variables (a través de una expresión aritmética o algebraica) a otra variable.

[5]Disponible en http://pseint.sourceforge.net/

Más adelante utilizaremos estructuras que representarán a bloques de selección o de repetición. Ahora representaremos de forma textual todos los problemas a resolver.

Para este primer ejercicio, se nota que el texto de pantalla 'Ingrese el radio: ' sugiere al usuario el ingreso de un valor, se ingresará el valor de 1, en el primer intento para conocer si está correcta la secuencia de pasos utilizada. Si usted observa la fórmula al multiplicar 1 por 1 nuevamente y por el valor de la variable *valor_PI* el resultado debe ser la misma variable *valor_PI* es decir 1 por 1 y por 3.1415 da como resultado 3.1415, con ello usted puede verificar que todos los pasos (líneas de código o instrucciones anteriores) fueron bien utilizadas y la variable *ra* no tuvo problemas con el ingreso por teclado del valor utilizado. Para la instrucción de Escribir en PSeInt, puede usarse comillas simples o dobles.

Algoritmo 1: Ejercicio01 -Área del círculo sin constante PI

```
Escribir 'Ingrese el radio: '
Leer ra
valor_PI← 3,1415
area ← valor_PI∗ra∗ra
Escribir 'Respuesta: ', area
```

Fin Algoritmo

La variable *ra* se le asignará el valor ingresado desde teclado. La variable *valor_PI* se inicializará el valor constante de 3.1415. En la variable area (sin acento, sin tilde) se asignará la expresión correspondiente a la fórmula del área del círculo, es el proceso central del algoritmo. Finalmente se presenta el valor almacenado en la variable area luego de una cadena de caracteres 'Respuesta:'.

Si mejoramos el algoritmo y utilizamos valores propios o determinados por defecto del mismo programa por ejemplo el valor de 3.1415 ya viene dado en la palabra PI, a esto se conoce como "palabras reservadas" de programa y PI ya tiene un valor que proviene de alguna librería del software de PSeInt.

Algoritmo 2: Ejercicio02 -Área del círculo con constante PI

```
Escribir 'Ingrese el radio: '
Leer ra
area ← PI∗ ra ∗ ra
Escribir 'Respuesta: ', area
```

Fin Algoritmo

Si ejecutamos el programa, e ingresamos el valor de 1, la respuesta sería de 3,141592653, valor que guarda PSeInt.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el área de un triángulo en función de las longitudes de sus lados. Para ello la fórmula del área es $area = \sqrt{p(p-a)(p-b)(p-c)}$ donde la variable p es $p = (a+b+c)/2$.

Para este ejercicio se debe observar que nos dan la fórmula (proceso) para hallar el área del triángulo. La variable de salida es *area* (sin tilde). Se observa que existe la variable p, que es una fórmula donde se suman otras tres variables (entradas) y luego se divide para dos. Habiendo entendido el problema ahora debemos determinar las variables de entrada que son a, b y c. Lo nuevo en este problema es la operación de la raíz cuadrada. Utilizaremos una función propia del programa PSeInt llamada raíz.

Algoritmo 3: Ejercicio03 -Área del triángulo en función de sus lados

Escribir 'Ingrese lado a: '
Leer a
Escribir 'Ingrese lado b: '
Leer b
Escribir 'Ingrese lado c: '
Leer c
p $\leftarrow (a+b+c)/2{,}0$
area $\leftarrow$ raiz $(p*(p-a)*(p-b)*(p-c))$
Escribir 'Resultado es: ', area

Fin Algoritmo

Si hacemos una revisión del algoritmo desde el programa del PSeInt tenemos el ejercicio03 en la siguiente figura (ver Fig. 13):

```
1  Algoritmo ejercicio03
2      Escribir "Ingrese lado a: "
3      Leer a
4      Escribir "Ingrese lado b: "
5      Leer b
6      Escribir "Ingrese lado c: "
7      Leer c
8      p ← (a+b+c)/2.0
9      area ← raiz(p*(p-a)*(p-b)*(p-c))
10     Escribir 'Resultado es: ', area
11 FinAlgoritmo
```

Figura 13: *Solución en PSeInt del problema del área del triángulo.*

Se puede notar que para cada ingreso de una variable de entrada (Leer) se debe colocar un mensaje previo (Escribir) de tal manera que el usuario comprenda qué debe ingresar con la correspondiente orden. Se observa el proceso de la fórmula para la variable p escrita de forma lineal con el uso correspondiente de paréntesis que determinan la expresión para el numerador y el denominador es un número real de 2.0, es importante comprender que un resultado puede esperarse los números

decimales necesarios o para otras respuestas no sean necesarios. Por eso es importante saber qué respuesta se desea obtener. Finalmente, la variable *area* (sin tilde ya que es un elemento del programa) tiene una función propia del programa llamada raíz, que obtiene un número real.

Al ingresar valores correspondientes para las variables a, b y c se obtiene un resultado en un número real (ver Fig. 14). Recuerde que, si ingresa valores que la fórmula no contemple mostrar y el programa genera un error, no significa que el algoritmo esté mal, sino que la respuesta no es permitida para ciertos valores de acuerdo con las reglas de las operaciones como la raíz cuadrada que no opera para números negativos.

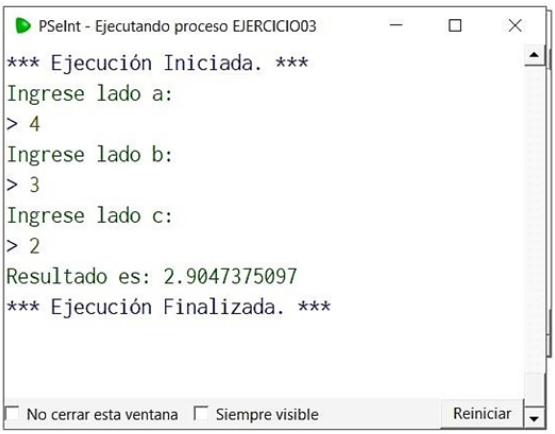


Figura 14: *Resultado de la ejecución del algoritmo en PSeInt.*

Finalmente, se muestra diagrama de flujo que genera el programa PSeInt (ver Fig. 15), se muestran los mensajes para cada una de las variables (paralelogramo en color verde), las variables de entrada (paralelogramo en color rosado) y los procesos (rectángulos de color amarillo).

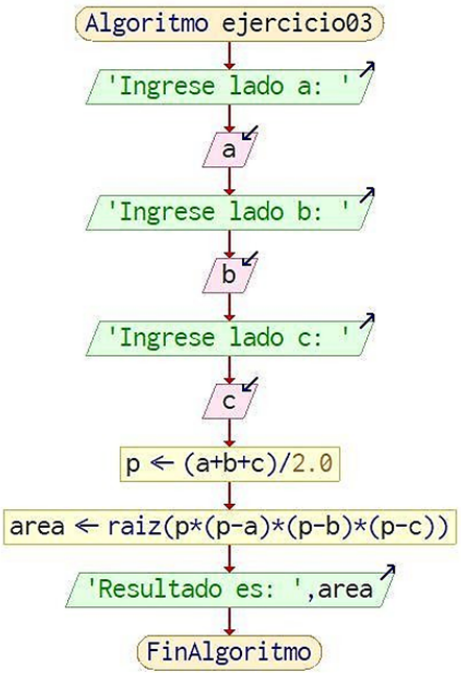


Figura 15: *Diagrama de flujo en PSeInt del ejercicio anterior.*

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el cambio de temperaturas de grados Celsius a grados Fahrenheit. Para ello la fórmula de la conversión apropiada es $f = \left(\frac{9{,}0}{5}\right) * c + 32$.

El algoritmo sería el siguiente:

Algoritmo 4: Ejercicio04 -Conversión de temperaturas

Escribir 'Ingrese $°C$: '
Leer c
f $\leftarrow (9{,}0/5) * c + 32$
Escribir 'Fahrenheit es: ', f

Fin Algoritmo

Se observa en la siguiente figura (ver Fig. 16) que, en el ejercicio04, el algoritmo utiliza una fórmula y esta debe ser expresada en una sola línea para que lo entienda el programa que ejecutará el algoritmo. Puede entenderse que hay programas que al no colocar decimales 9.0 todo lo va a interpretar como enteros, inclusive la división entera (que existe ejemplo 9/5 daría 4 y no 4.5 por no encontrar decimales, el 9 y el 5 son enteros, 9.0 es decimal).

```
1 Algoritmo ejercicio04
2     Escribir "Ingrese °C"
3     Leer c
4     f<-(9.0/5) * c + 32
5     Escribir "Fahrenheit es",f
6 FinAlgoritmo
```

Figura 16: *Pseudocódigo del algoritmo conversión de temperaturas.*

Podemos entender de este programa que se pide al usuario ingresar un valor en $°C$, luego el programa (algoritmo) leerá lo que ingrese o digite, para realizar como proceso lo que se almacena en f y finalmente escribir le mensaje final con el valor de f.

En la figura 17 se puede observar el diagrama de flujo del algoritmo conversión de temperaturas. En los diagramas de flujo se evidencia la diferencia la entrada y salida con los paralelogramos y unas flechas en la parte superior derecha hacia afuera y hacia adentro.

Al ejecutar el algoritmo en PSeInt, el resultado se muestra en la figura 18, podemos observar que el usuario (usted ahora) ha ingresado el valor del 23. Y el algoritmo desarrollado por el programador (usted antes) lo ha hecho bien, ya que el resultado de la fórmula utilizada da 73.4 grados Fahrenheit.

En otro ejemplo de conversiones, puedas notar la escritura de una fórmula en una sola línea.

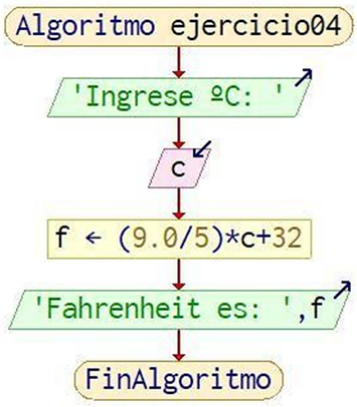


Figura 17: *Diagrama de flujo del algoritmo conversión de temperaturas.*

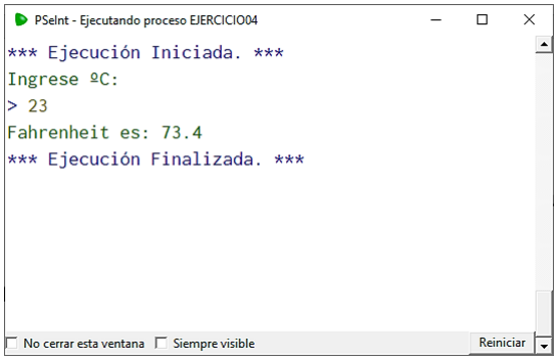


Figura 18: *Resultado del algoritmo conversión de temperaturas.*

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el cambio de un ángulo en grados a radianes. Para ello la fórmula de la conversión apropiada es *radianes*$= \left(\frac{\text{Ángulo}}{180}\right) * \pi$, recuerde que $360°$ es 2π *radianes*, que equivale a 2 por $3{,}141592$ (valor de π de 6 decimales) dando $6{,}283184$ (aproximadamente).

Algoritmo 5: Ejercicio05 -Conversión de Ángulos a Radianes

Escribir 'Ingrese un ángulo (número): '
Leer *angu*
radianes $\leftarrow (angu/180) * PI$
Escribir 'El valor del ángulo ', angu, ' en radianes es: ', radianes

Fin Algoritmo

Se aprecia en el algoritmo 5 que el texto de "Ingrese un ángulo (número):" se muestra usando acento en la palabra ángulo y paréntesis entre la palabra número, es decir que al ser un texto que se visualiza en pantalla, la función Escribir refleja el texto escrito en el programa. El uso correcto de signos de admiración, exclamación y acentos debe ser bien empleado para que el usuario entienda la orden que debe realizar.

En la figura que se muestra a continuación (ver Fig. 19), en el ejercicio05 se usan dos barras inclinadas (división seguidas) representando un comentario "//se puede usar comillas dobles", como una ayuda al lector-programador que se encuentra en el desarrollo del programa.

```
1 Algoritmo ejercicio05
2     Escribir "Ingrese un ángulo(un número):" //se puede usar comillas
3     Leer angu
4     radianes ← (angu / 180) * PI
5     Escribir "El valor del ángulo  ", angu, ' en radianes es: ', radianes
6 FinAlgoritmo
```

Figura 19: *Pseudocódigo del algoritmo conversión de grados a radianes.*

Finalmente, el resultado (ver Fig. 20) del algoritmo, si ingresamos el valor de 360 en la ejecución del programa nos da el mismo valor si hubiéramos utilizado una calculadora. Y ese es el fin de realizar algoritmos, que mediante nuestro propio ingenio y creatividad podamos crear nuevos programas que permitan realizar cómputos y ayudarnos en nuestras actividades diarias o estudio.

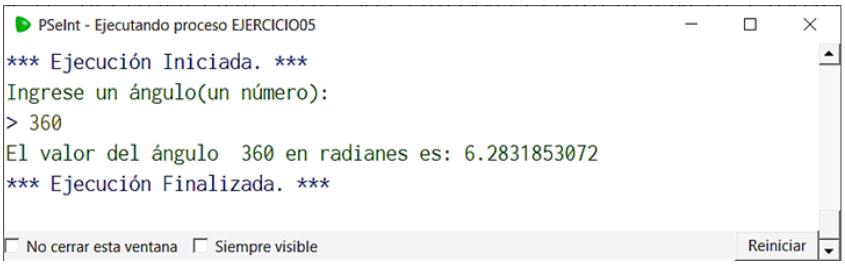


Figura 20: *Ejecución del algoritmo conversión de grados a radianes.*

Si observamos el diagrama de flujo del algoritmo (ver Fig. 21), la representación de paralelogramos permite distinguir las entradas y salidas. Mientras que la representación del rectángulo permite distinguir los procesos como por ejemplo las fórmulas.

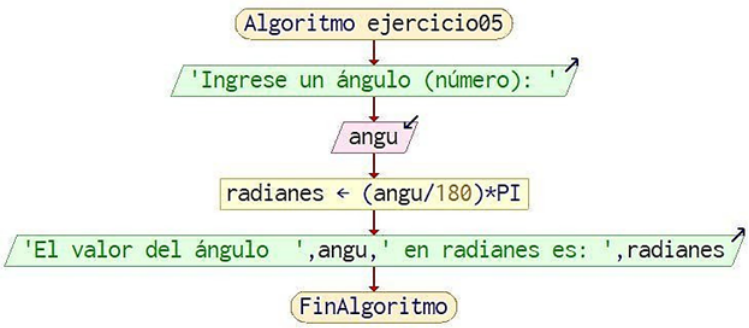


Figura 21: *Diagrama de flujo del algoritmo de conversión de grados a radianes.*

Estructuras selectivas simples

Para las estructuras selectivas, debes recordar que son como un flujo que encuentra una bifurcación, es decir nos toparemos con una condición donde se debe escoger entre dos caminos, por verdadero y por falso. En este caso al ser simple es más restrictiva ya que solo está el flujo por verdadero, por falso no se preguntará ni se realizará nada, observa.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea ingresar un número por teclado y el algoritmo debe mostrar en pantalla si es mayor cero (para menores e iguales a cero no se realizada nada).

El algoritmo es el siguiente:

Algoritmo 6: Ejercicio06 -Verificar si un número ingresado es mayor a cero

Escribir Ïngrese un número://se puede usar comillas
Leer *num*
Si num >0 Entonces
    Escribir *num*, “es mayor a cero”
//SiNo
    //acciones_por_falso
Fin Si

Fin Algoritmo

Se observa que el software de PSeInt permite el uso de comillas simples como dobles. A partir de este momento usaremos las dobles. Aparece la estructura selectiva simple “Si” luego la condición “num>0” y la acción por verdadero que es escribir el mensaje. También se aprecia que la acción por falso, para este ejemplo, se encuentra en comentarios (con dos barras inclinadas, llamadas también división, operador cociente) es decir, están ocultas para la ejecución del programa.

En la figura que se presenta a continuación (ver Fig. 22) se muestra el ejercicio06 y aquellas palabras que utiliza el software para ejecutar el algoritmo con texto de color. Y en tonos grises aquello que podemos ver, pero se encuentra oculto a la ejecución del algoritmo. Tantos comentarios podemos colocar dentro del algoritmo, de tal forma que ayuden al programador o uno mismo a entender por qué de las líneas utilizadas.

```
1 Algoritmo ejercicio06
2     Escribir "Ingrese un número:" //se puede usar comillas
3     Leer num
4     Si num > 0 Entonces
5         Escribir num, " es mayor a cero"
6     //SiNo
7         //acciones_por_falso
8     Fin Si
9 FinAlgoritmo
```

Figura 22: *Pseudocódigo del algoritmo si un número ingresado es mayor a cero.*

Puede servir como una buena práctica, cuando estamos pensando en la lógica del algoritmo escribirlo como comentario así, algo que nos queda pendiente lo vamos apuntando o que hayamos descubierto y que funciona para nuestro algoritmo o para otro, podamos recordarlo en otro momento y volverlo a utilizar (ver Fig. 23).

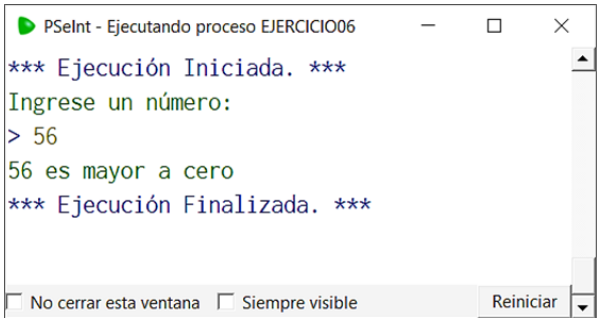


Figura 23: *Ejecución del algoritmo si un número ingresado es mayor a cero.*

La imagen a continuación (ver Fig. 24), presenta el diagrama de flujo del algoritmo anterior, debemos observar que se muestran los comentarios colocados, pero no afectan la ejecución del algoritmo. La utilidad de los comentarios es de alta importancia cuando se tengan varias líneas de código que superen altos valores, como de miles de líneas y que esté bien documentadas. Use continuamente comentarios y verá su utilidad cuando regrese al algoritmo luego de mucho tiempo.

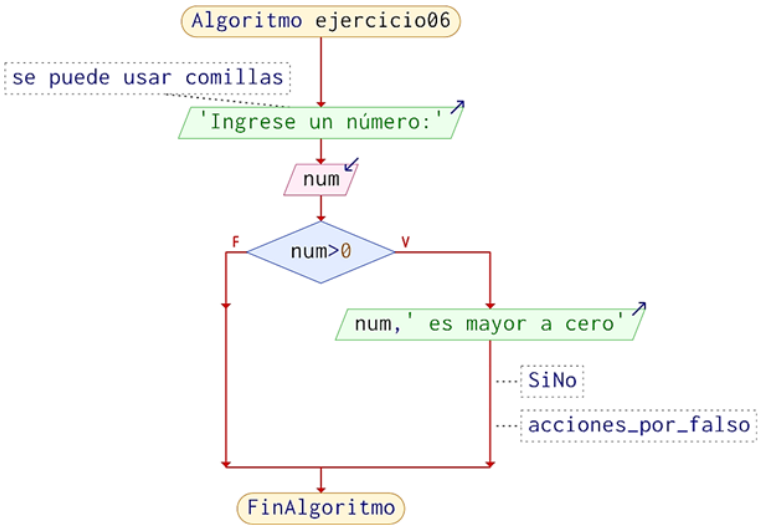


Figura 24: *Diagrama de flujo si un número ingresado es mayor a cero.*

En la siguiente sección utilizaremos estructuras selectivas dobles.

Estructuras selectivas dobles

Para las estructuras selectivas dobles, debemos comprender que aquí se toman en cuentan dos bifurcaciones, es decir cuando se llega a la condición se debe analizar qué acciones irán cuando la condición es verdadera y qué acciones irán cuando la condición es falsa.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea ingresar un número entero por teclado y el algoritmo debe determinar si es par o impar, para esto se muestra en pantalla un correspondiente mensaje: “2 es par” o “75 es impar”.

El algoritmo es el siguiente:

Algoritmo 7: Ejercicio07 -Verificar si un número ingresado es para o impar

```
Escribir Ïngrese un número entero: "
Leer nume
Si (nume MOD 2 = 0) Entonces
        Escribir nume, “es par”
SiNo
        Escribir nume, “es impar”
Fin Si
```

Fin Algoritmo

Si observamos el seudocódigo anterior que aparece una palabra nueva “MOD”, esta palabra es una ***palabra reservada*** del programa, es decir, todos los programas, utilitarios o no, como Word, Excel, así como los lenguajes Java, Python, R, entre otros, tienen palabra que se reservan para un uso exclusivo. MOD representa el módulo o resto.

Si revisas en el software de PSeInt la ayuda en el menú (ver Fig. 25), puedes encontrar información relevante para comprender las posibles combinaciones y usos de las palabras reservadas, así como cuando usamos el valor de la constante PI en un ejercicio anterior.

Figura 25: *Opciones del menú del software de PSeInt.*

De esta opción del menú encontramos una expresión a utilizar y su significado, en la tabla 17 colocamos el resumen de las expresiones en lenguaje formal.
Donde se debe considerar lo que se conoce como jerarquía de expresiones que toman en cuenta a la jerarquía de los operadores matemáticos que conocemos. Un ejemplo es, “M+13 ES IGUAL A N*4” equivale decir o entender a “$(M + 13) = (N * 4)$”. De igual manera, se puede determinar la condición opuesta de la expresión utilizando "NO ES". Para este ejemplo puede observar, “M NO

ES IGUAL A N” equivale expresar de manera formal que “NO (M=N)”.

Tabla 17: *Expresiones en lenguaje formal y su descripción.*

Expresión	Significado
X ES Y	X=Y
X ES IGUAL A Y	X=Y
X ES DISTINTO DE Y	X<>Y
X ES MAYOR QUE Y	X>Y
X ES MENOR QUE Y	X<Y
X ES MAYOR O IGUAL A Y	X>=Y
X ES MENOR O IGUAL A Y	X<=Y
X ES CERO	X=0
X ES POSITIVO	X>0
X ES NEGATIVO	X<0
X ES PAR	X MOD 2 = 0
X ES IMPAR	X MOD 2 = 1
X ES MULTIPLO DE Y	X MOD Y = 0
X ES DIVISIBLE POR Y	X MOD Y = 0

Nota: Información tomada del software de PSeInt

De la imagen a continuación (ver Fig. 26), el ejercicio07 muestra un ejemplo con la palabra reservada “MOD”, y el significado de la condición en la línea 4, nos indica que si un valor almacenado en nume, al realizar la división entera con 2, si su resultado es cero, entonces se escribirá “nume es par”, caso contrario “nume es impar”. Si nos detenemos y pensamos que si esto es cierto (por verdadero), tendremos un número par realmente. Divide mentalmente 12 o 48 para dos, la respuesta es exacta (cero en residuo), dará 6 o 24 (respectivamente) y el residuo será cero, es decir la división entera nos permite acoger la respuesta del residuo. En cambio, si tenemos 13 o 49, la respuesta de MOD 2 = 0 será uno, que significa que al dividir 13 o 49 tendremos también 6 y 24 (respectivamente) pero el residuo será uno, para todo número impar.

```
1 Algoritmo ejercicio07
2     Escribir "Ingrese un número entero: "
3     Leer nume
4     Si (nume MOD 2 = 0) Entonces
5         Escribir nume, "es par"
6     SiNo
7         Escribir nume, "es impar"
8     Fin Si
9 FinAlgoritmo
```

Figura 26: *Pseudocódigo del algoritmo si un número ingresado es par o impar.*

En la imagen que se muestra a continuación (ver Fig. 26) podemos observar el procedimiento que realiza MOD, la división entera, cuando ingresamos dos números, uno par (3956) y otro impar (4311). Cabe recordar que los números que ingresamos al programa serán los dividendos, el 2 será el divisor, y el cociente, aunque se lo obtenga no lo usaremos, el resultado que *nume* MOD 2 = 0 obtiene, es el residuo.

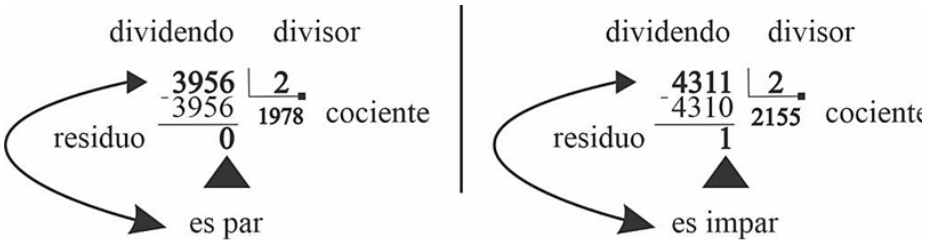


Figura 27: *Proceso que realiza MOD, la división entera.*

Si ejecutamos el algoritmo y colocamos los números de ejemplo de la figura 27 tenemos:

Figura 28: *Resultado del algoritmo ingresando 3956.*

En la Figura 28, se muestra el resultado “es par”, mientras que en la figura 29, tenemos el resultado de “es impar”, de acuerdo con los valores ingresados.

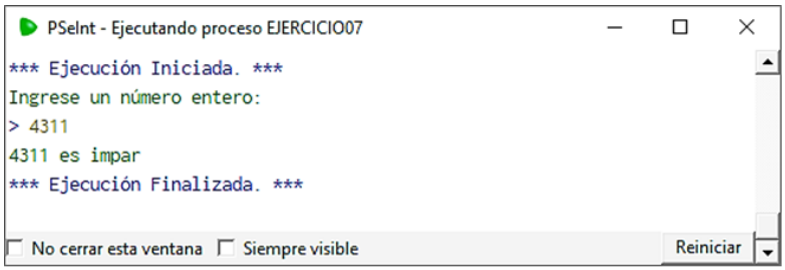


Figura 29: *Resultado del algoritmo ingresando 4311.*

El diagrama de flujo (ver Fig. 30) muestra las instrucciones por verdadero (derecha) y por falso (izquierda) de acuerdo con el resultado de la condición (rombo).

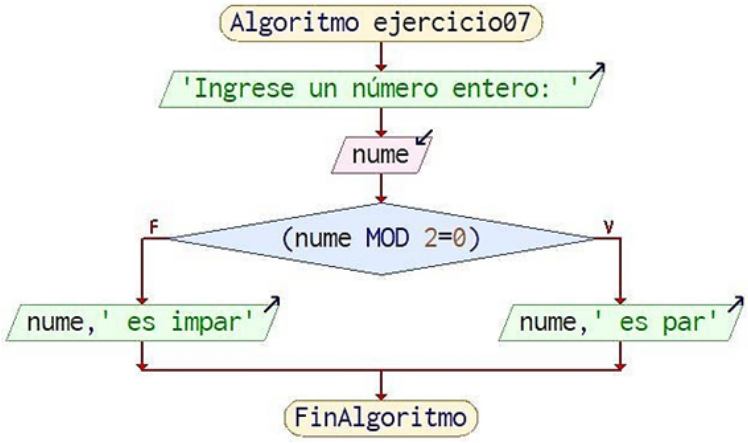


Figura 30: *Diagrama de flujo del ejercicio ingrese un número entero.*

Estructuras selectivas múltiples

Cuando se tienen más de un camino o flujo y la condición permita varias opciones, decimos que la estructura es de selección múltiple, veamos el siguiente ejemplo.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se ingresa por teclado un número del 1 al 4, el algoritmo debe mostrar una estación, primavera, verano, otoño e invierno, respectivamente.

El algoritmo es el siguiente:

Algoritmo 8: Ejercicio08 -Estaciones del año

```
Escribir Ïngrese un número entero: "
Leer nu
Segun nu Hacer
        1:
                Escribir “Primavera”
        2:
                Escribir “Verano”
        3:
                Escribir “Otoño”
        4:
                Escribir “Invierno”
        De Otro Modo:
                Escribir “Ingreso no corresponde del 1 al 4”
Fin Segun
```

Fin Algoritmo

Si observamos el seudocódigo anterior aparece una nueva estructura llamada “Según” y sea la opción (valor) ingresada se procede a seguir el flujo por el caso (valor numérico) que acompaña a los dos puntos, luego se ejecutan la línea o líneas que se hayan agregado en dicha opción.

Nota que en el ejercicio08 (ver Fig. 31), el algoritmo solicita al usuario que ingrese un número entero, luego el algoritmo lee el número y lo almacena en la variable *nu*, para luego llegar a una condición de bifurcación múltiple conocida como “Según hacer”. Las diferentes opciones, que pueden ser diferentes tipos de datos (enteros, reales, alfanuméricos) seguido de los dos puntos, permiten bifurcar la ejecución dependiendo si cumple con el valor en la opción esperada. Es decir, si se ingresa 1, como valor digitado por el usuario, se muestra “Primavera”, de igual forma, se ingresa 2, como valor, se muestra “Verano”, así, el 3, se muestra “Otoño” y finalmente si se ingres el 4, se muestra el mensaje de “Invierno”. Recuerda que pueden ser una o más líneas de instrucción.

Para el caso de que no se encuentre el valor en ninguno de los casos (en este ejemplo del 1 al 4), por ejemplo, el 5 o el -4, irán a la opción por defecto conocida como “De Otro Modo”, con ello

podemos validar el ingreso, esperando que el usuario no se equivoque al ingresar otro número que no sea solicitado.

```
1  Algoritmo ejercicio08
2      Escribir "Ingrese un número entero: "
3      Leer nu
4      Segun nu Hacer
5          1:
6              Escribir "Primavera"
7          2:
8              Escribir "Verano"
9          3:
10             Escribir "Otoño"
11         4:
12             Escribir "Invierno"
13         De Otro Modo:
14             Escribir "Ingreso no corresponde del 1 al 4"
15     Fin Segun
16 FinAlgoritmo
```

Figura 31: *Pseudocódigo del ejercicio de estaciones del año.*

En este momento se muestra el resultado en la figura 32, y observamos que al haber ingresado el 3, se muestra el mensaje de “Otoño”.

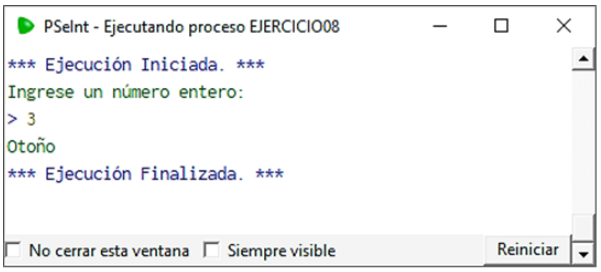


Figura 32: *Mensaje del algoritmo de la estación del año al ingresar 3.*

Para otro valor por ejemplo el 5, en la figura 33, se muestra el mensaje que se ha colocado, indicando al usuario que debe ingresar sólo lo que se le pide. Recordemos que el usuario al no tener un mensaje adecuado (como lo hicimos en este ejemplo al principio), el usuario hará lo que le pidamos (¡no siempre!) y por eso se recomienda, mensajes claros y precisos.

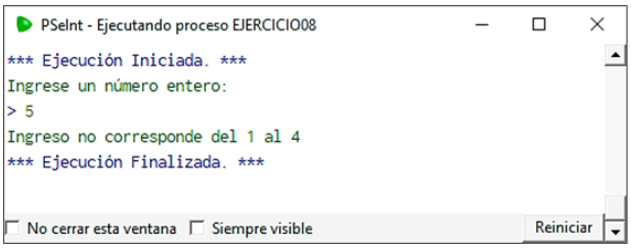


Figura 33: *Mensaje del algoritmo de la estación del año al ingresar 5.*

En cambio, si observamos el diagrama de flujo (ver Fig. 34), podemos notar que luego del ingreso del valor por teclado, la condición tiene varias alternativas de flujo a seguir.
Cuando utilicemos un lenguaje de programación esta estructura puede existir con otro nombre o puede representarse como la anidación de la estructura de selección doble y simple que vimos anterior a esta estructura.

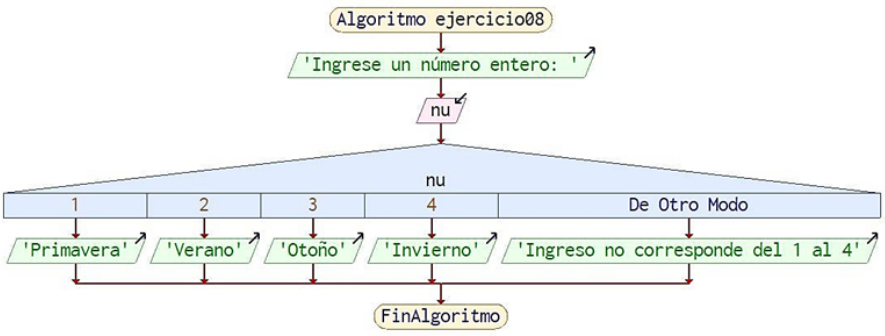


Figura 34: *Flujograma del algoritmo de estación del año.*

Con esto hemos superado las estructuras selectivas o de selección, puesto que siguen un camino o flujo, condicionadas en la bifurcación (rombo) y presentan la líneas o líneas que están la opción escogida. Ahora pasaremos a estructuras que se conocen como repetitivas debido a que son un lazo de repeticiones de una o más líneas o bloques de ellas. Estas estructuras repiten instrucciones que dependen de una condición de finalización. Estas condiciones generalmente utilizan operadores de relación, lógicos o booleanos. Veremos tres estructuras existentes en PSeInt, pero en los lenguajes de programación pueden reducirse.

Estructuras repetitivas mientras

La estructura repetitiva *mientras* o lazo *mientras*, ejecuta el bloque de líneas mientras la condición sea verdadera.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se requiere mostrar y sumar la serie de números de 11 al 21.

El algoritmo es el siguiente:

Algoritmo 9: Ejercicio09 -Sumar serie

```
i ← 11
sumar ← 0
Mientras i <= 21 Hacer
    Escribir i
    sumar ← sumar + i
    i ← i + 1
Fin Mientras
Escribir "El resultado es", sumar
```

Fin Algoritmo

Si observamos el seudocódigo anterior aparece una nueva estructura llamada "Mientras", y forma parte del conjunto de estructuras repetitivas con "Repetir" y "Para". Toda estructura repetitiva tiene tres partes lógicas que debes recordar, primero, la variable de inicialización, es $i \leftarrow 11$, su valor para este ejemplo es de once, pero si lo cambias a 1, empezará desde uno; segundo, la condición de finalización, es $i <= 21$, que se encuentra al inicio de la estructura, este valor indica que termina en 21 inclusive, es decir que va de 11 al 21 ya que la estructura Mientras repite por verdadero, y con esto se lee "mientras i sea menor e igual a 21 repite el bloque que viene a continuación «puedes notar que el bloque interior del lazo mientras está con tabulación/sangría/identado»"; tercero, el incremento/decremento, es $\leftarrow i + 1$, con ello la estructura ya puede iterar/repetir (puedes usar cualquiera de estos dos nombres).

Piensa por un momento y observa, si queremos que itere del 1 al 10, ¿qué debemos cambiar?, debe ser en dos lugares, la variable de inicialización con $i \leftarrow 1$ y la condición con $i <= 10$, ¡intenta modificar!

En la figura 35, puedes observar el contenido del bloque, recuerda que el problema es de encontrar la sumatoria del 11 al 21, por ende, la variable "sumar" se encera o se le asigna el valor de cero (línea 3). Dentro de la estructura repetitiva en la línea 6, se muestra la variable "sumar" incrementando el valor de i, esto significa que se suma cero más once y continúa de esa manera hasta llegar al 21.

```
1  Algoritmo ejerccicio09
2      i←11
3      sumar ← 0
4      Mientras i ≤ 21 Hacer
5          Escribir i
6          sumar ← sumar + i
7          i← i + 1
8      Fin Mientras
9      Escribir "El resultado es ",sumar
10 FinAlgoritmo
```

Figura 35: *Pseudocódigo del ejercicio de la sumatoria del 11 al 21.*

Luego de que finalice el algoritmo, se presenta la línea 9, donde se muestra el resultado de la variable sumar (y del problema de sumatoria), (ver Fig. 36).

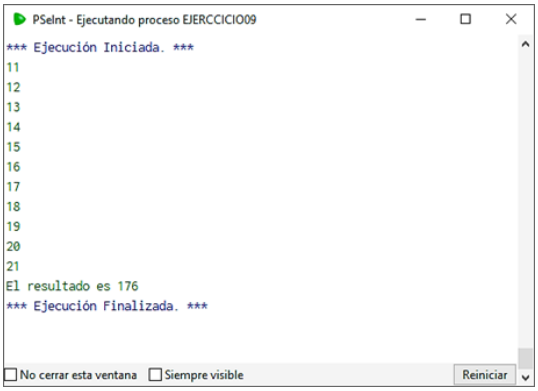


Figura 36: *Mensaje del algoritmo de la sumatoria del 11 al 21.*

En la figura 36, se muestra el mensaje de la sumatoria en cada iteración, al final de la ejecución se muestra el resultado de sumar 11 más 12, más 13, hasta el valor de 21.

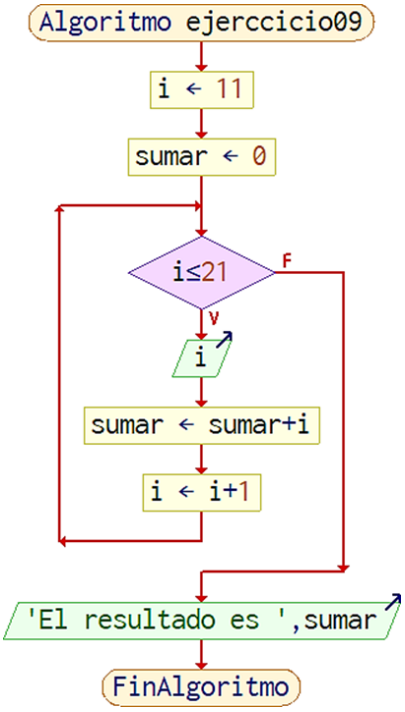


Figura 37: *Flujograma del algoritmo de la sumatoria del 11 al 21.*

Como puedes observar en la figura 37, en flujograma la estructura repetitiva “mientras” tiene un rombo en su parte superior (antes de entrar al bloque de código) eso significa que esta estructura es muy estricta, es decir no deja “pasar” o “ejecutar” su contenido a menos de que se cumpla la condición y esta sea verdadera.

Estructuras repetitivas *mientras* controladas por un contador

La estructura repetitiva *mientras* o lazo *mientras*, puede tener dos variantes. La primera es que sus repeticiones son controladas por un contador y la segunda, controladas por un *centinela*. Para un ejemplo de una estructura mientras controlada por contador sería lo siguiente.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Un profesor desea ingresa tres notas de sus alumnos de una asignatura, las notas son sobre veinte puntos.

El algoritmo es el siguiente:

Algoritmo 10: Ejercicio10 -Ingreso de tres notas

```
contador = 1
acumulador = 0
Mientras contador <= 3 Hacer
    Escribir “Ingreso de tres notas”
    Leer nota
    acumulador ← acumulador + nota
    contador ← contador + 1
FinMientras
promedio ← acumulador / (contador −1)
Escribir ’El promedio de las tres notas es ’, promedio
```

Fin Algoritmo

Si observamos el seudocódigo anterior aparece una nueva estructura llamada “Mientras”, similar al ejercicio anterior, entonces ¿en qué cambia?, es el contador, una variable que puede tener varios nombres, variable de control o variable de iteración, pero que en definitiva determina la cantidad contada de ingresos que requieres, en este caso son las tres notas. En la figura 38, debes observar el flujo que sigue el contador en las líneas, 2, 4 y 8. Si observas esta variable llega hasta contar 4, por eso sale del lazo luego del 3 y en la línea 10 se la requiere para obtener el promedio, pero al ser 4, se resta a su contenido 1, para promediar para 3. A este tipo de algoritmo se lo conoce como estructura mientras controlado por contador.

```
1  Algoritmo ejerccicio10
2      contador = 1
3      acumulador = 0
4      Mientras contador ≤ 3 Hacer
5          Escribir "Ingreso de tres notas"
6          Leer nota
7          acumulador ← acumulador + nota
8          contador ← contador + 1
9      FinMientras
10     promedio ← acumulador / (contador - 1)
11     Escribir 'El promedio de las tres notas es ',promedio
12 FinAlgoritmo
```

Figura 38: *Pseudocódigo del ejercicio del ingreso de tres notas.*

Si ejecutamos el seudocódigo anterior (ver Fig. 38), notamos que el resultado en la ventana (ver Fig. 39) permite el ingreso de tres notas y obtiene el promedio.

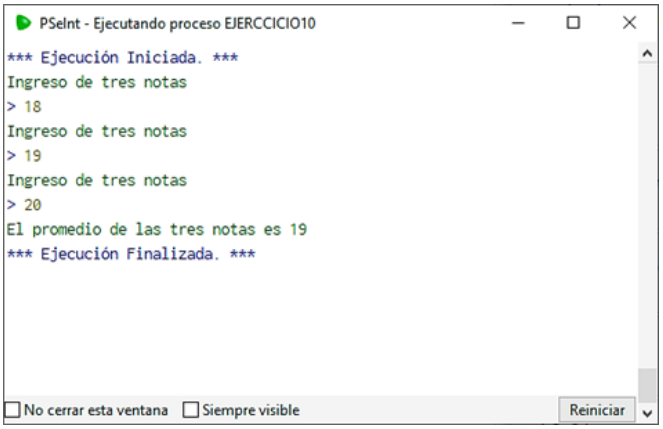


Figura 39: *Mensaje del algoritmo del ingreso de tres notas.*

La figura 40, se muestra el diagrama de flujo que presenta la solución del problema.

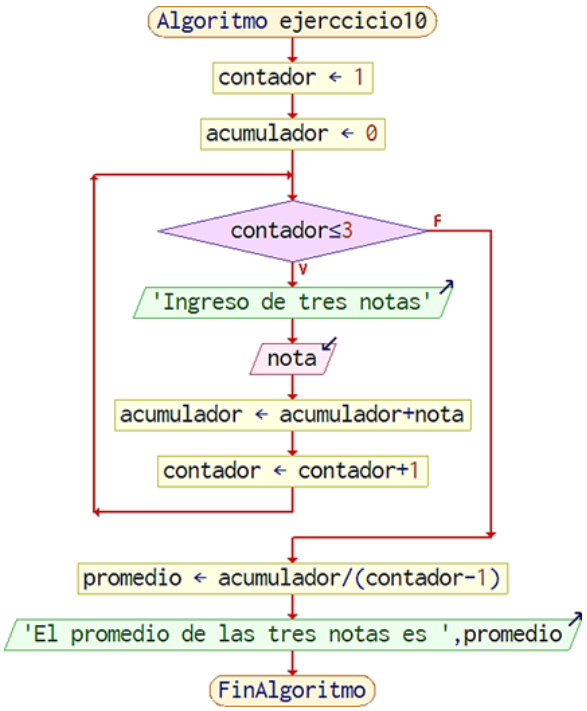


Figura 40: *Flujograma del algoritmo del ingreso de tres notas.*

Estructuras repetitivas *mientras* controladas por un centinela

La estructura repetitiva mientras o lazo mientras, al ser controlada por un centinela, significa que los valores de entrada que valida de entre ellos, uno es totalmente diferente a los demás o hacemos que valide a ese valor de forma diferente. El ejemplo sería cuando ingresamos “notas” de estudiantes (ya no un determinado número sino varios de forma indeterminada), al no saber cuántos, podemos indicar al algoritmo que si se ingresa una nota (sobre veinte, por ejemplo) de un valor de -99 (valor sobre veinte que no sería válido para considerarlo como nota) el algoritmo finalizaría.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Un profesor desea ingresar **varias** notas de sus alumnos de una asignatura, las notas son sobre veinte puntos. Si se ingresa el valor de -99, el algoritmo finalizaría y presentaría cuántas notas ha ingresado sin contar el último valor (el centinela).

El algoritmo es el siguiente:

Algoritmo 11: Ejercicio11 -Ingreso de varias notas

```
contador = 0
acumulador = 0
Escribir Ïngreso de notas"
Leer nota
Mientras nota <> −99 Hacer
    acumulador = acumulador + nota
    contador = contador + 1
    Escribir Ïngreso de notas"
    Leer nota
FinMientras
Si acumulador <> 0 Entonces
    promedio = acumulador / (contador)
    Escribir ’El promedio de ", contador,"notas es ’, promedio
SiNo
    Escribir "No se ha ingresado notas aún"
Fin Si
```

Fin Algoritmo

Como podemos observar en el algoritmo anterior la variable *centinela* es “nota” debido a que por medio de ella todas las calificaciones serán ingresadas y acumuladas, pero si al ingresar, aunque sea la primera vez el valor del -99, la estructura repetitiva *mientras* lo va a impedir o detener. Se ingresarán varias notas, un número indeterminado, pero si luego de la primera ejecución válida se ingresa el valor de -99, entonces finalizará el algoritmo.

Si ejecutamos el algoritmo (ver Fig. 41), podemos identificar que en las líneas 12 y 15 se presentan un conjunto de sentencias en el caso de que haya aumentado el acumulador, también se hubiera validado la variable “contador”, ya que si aumenta el acumulador (o el contador) significaría que ha habido un buen ingreso, caso contrario se muestra el mensaje de “No se ha ingresado notas aún”.

```
1  Algoritmo ejercicio11
2      contador = 0
3      acumulador = 0
4      Escribir "Ingreso de notas"
5      Leer nota
6      Mientras nota ≠ -99 Hacer
7          acumulador ← acumulador + nota
8          contador ← contador + 1
9          Escribir "Ingreso de notas"
10         Leer nota
11     FinMientras
12     Si acumulador ≠ 0 Entonces
13         promedio ← acumulador / (contador)
14         Escribir 'El promedio de ", contador," notas es ', promedio
15     SiNo
16         Escribir "No se ha ingresado notas aún"
17     Fin Si
18 FinAlgoritmo
```

Figura 41: *Pseudocódigo del ejercicio del ingreso de varias notas.*

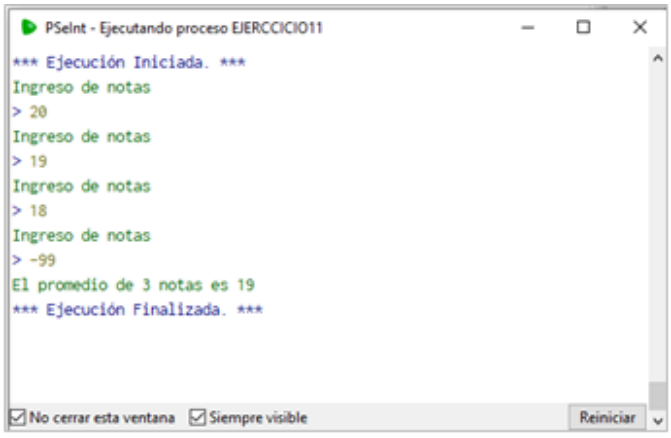


Figura 42: *Mensaje del algoritmo del ingreso de varias notas con tres ingresos.*

El mensaje que se muestra del algoritmo es visible en la figura 42, recuerda que se han ingresado tres notas (pudiera haber sido muchas más) y contador llega a tres para dividir al acumulador.

Si volvemos a ejecutar el algoritmo, pero en esta ocasión, en el primer ingreso colocamos -99, entonces se muestra el mensaje por falso debido a que fue -99 el que ingresó y no hubo nota alguna, ni contador incrementado, ni acumulador aumentado.

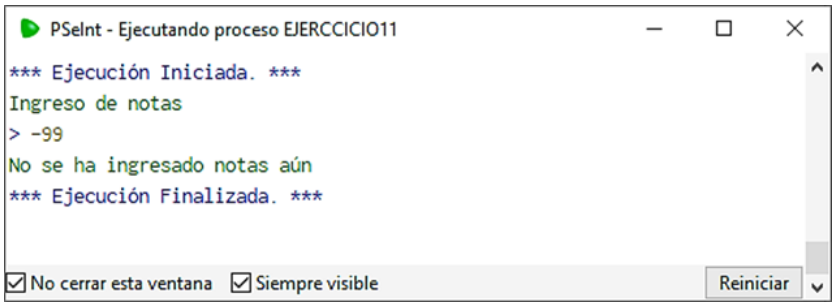


Figura 43: *Mensaje del algoritmo del ingreso de varias notas con -99 en el primer ingreso.*

Podemos observar (ver Fig. 43), el mensaje que se produce luego de que iniciado el algoritmo se ingresa el -99.

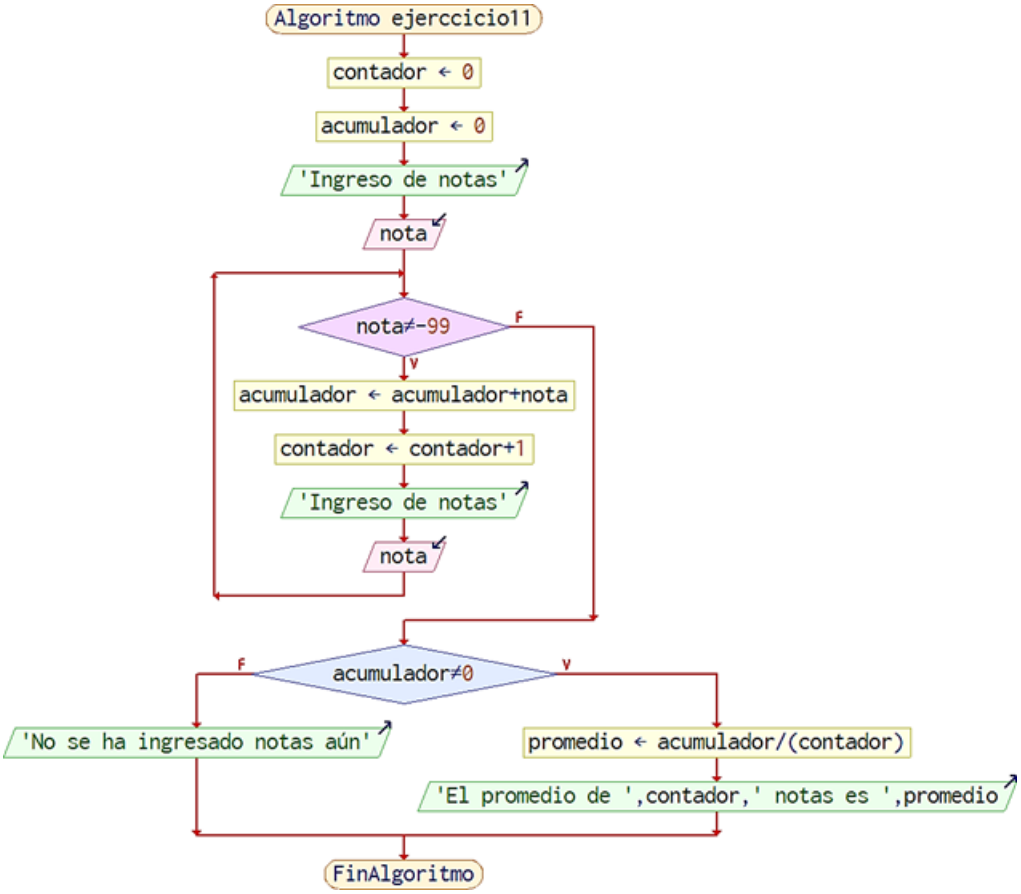


Figura 44: *Flujograma del algoritmo del ingreso de varias notas con detección del centinela.*

Estructuras repetitivas repetir hasta

La estructura repetitiva repetir hasta o lazo hasta, ejecuta el bloque de líneas hasta que la condición sea verdadera (es decir se mantiene en falso). Esta estructura con el tiempo pasará a repetir por verdadera como el mientras y luego en los lenguajes de programación desaparecerá, trata de identificar lo que indicamos al inicio de este párrafo por ahora, "hasta que la condición se cumpla".

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea presentar los números del 11 al 21 de forma descendente.

El algoritmo es el siguiente:

Algoritmo 12: Ejercicio12 -Números descendentes

num = 11
Repetir
    Escribir num
    num = *num* +1
Hasta Que num > 21

Fin Algoritmo

En el algoritmo anterior notamos que la variable *num* se inicializa en 11, al ingresar a la estructura *repetir* muestra el valor del contenido y en la siguiente línea incrementa en 1. La condición que controla a la estructura indica hasta que el valor de *num* sea mayor a 21, si esto no se cumple, por falso, repite. Es decir, esta estructura se queda en el bucle de repeticiones, hasta que num de 11 logre llegar y superar el valor de 21.

```
1 Algoritmo ejercicio12
2     num = 11
3     Repetir
4         Escribir num
5         num = num + 1
6     Hasta Que num > 21
7 FinAlgoritmo
```

Figura 45: *Pseudocódigo del uso de la estructura repetir.*

Trata de entender y comparar entre las estructuras *mientras* y *repetir* cuando usan los operadores de relación, una usa menor y la otra mayor, no es una regla, sino que debes pensar de forma lógica para la aplicabilidad del operador adecuado. Por eso debes aplicarte a ti la lógica como que, si tú fueras el algoritmo y sabrás qué operador utilizar, (ver Fig. 45).

Si ejecutamos el algoritmo podremos observar la presentación de la serie de números del 11 al 21 tal como nos pide el problema. Y puede ser escrito de diferentes maneras que te presentaremos a continuación, (ver Fig. 46).

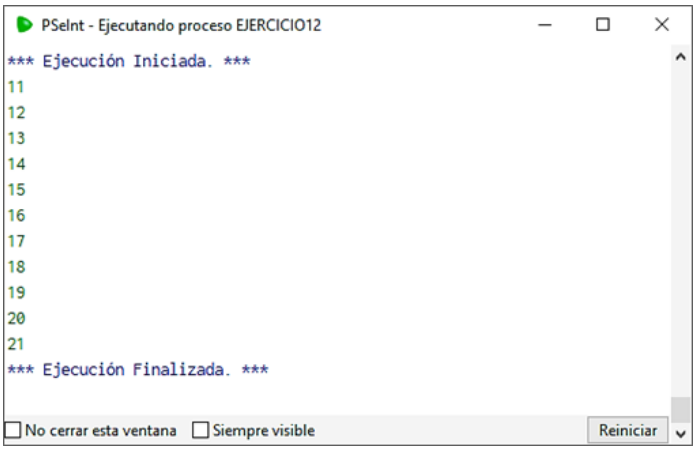


Figura 46: *Mensaje de salida del algoritmo con el uso de la estructura repetir.*

El diagrama de flujo que presentamos a continuación, se muestra la variable *num* inicializada en 11, con un gráfico de un rectángulo y una flecha hacia la izquierda, (ver Fig. 47). Recuerda que los rectángulos en los diagramas de flujo son asignaciones (el valor que se les asigna o contienen) o son procesos (fórmulas para asignar). Las entradas o salidas de datos se dibujan con un paralelogramo. De igual manera las condiciones con un rombo. Es sencillo observar el flujo de líneas que te indican que debes seguir (con tu imaginación) aplicando la lógica algorítmica, secuencial o no.

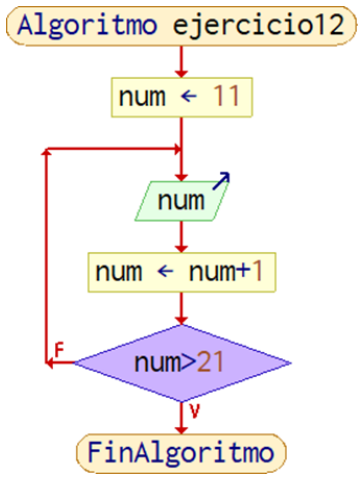


Figura 47: *Flujograma del algoritmo del uso de la estructura repetir.*

Como se observa en la figura 47, la condición (representada por un rombo) muestra que si la condición es falsa (una F) se repite el ciclo, caso contrario sale del lazo o bucle con verdadero (una V).

¿Pero es posible escribir un mismo algoritmo de formas distintas?, y la respuesta es sí. A continuación, colocaremos algunos algoritmos donde la salida es la misma pero las estructuras y las condiciones varían, así como sus diagramas de flujo.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea presentar los números del 11 al 21 de forma descendente, (versión 2).

El algoritmo de esta versión 2, es el siguiente:

Algoritmo 13: Ejercicio13 -Números descendentes versión 2

num = 10
Repetir
    num = *num* +1
    Escribir *num*
Hasta Que *num* > 20

Fin Algoritmo

En el algoritmo anterior notamos que la variable *num* se inicializa en 11, pero ahora la variable *num* se inicializa en 10 y el valor de la condición se deja en 20. Es decir, se ha procedido a modificarse los límites. Adicionalmente, puedes observar que las líneas del incremento y de presentación de la variable se encuentran en otro orden que permite la presentación después del incremento, si se deja con el orden del algoritmo anterior se presentaría el valor del 10, que no queremos, y para esto se corrige intercambiando de lugar las líneas de las sentencias, (ver Fig. 48).

```
1 Algoritmo ejercicio13
2     num ← 10
3     Repetir
4         num ← num+1
5         Escribir num
6     Hasta Que num > 20
7 FinAlgoritmo
```

Figura 48: *Pseudocódigo del uso de la estructura repetir segunda versión.*

En el diagrama de flujo, (ver Fig. 49), presenta el nuevo orden de las sentencias.

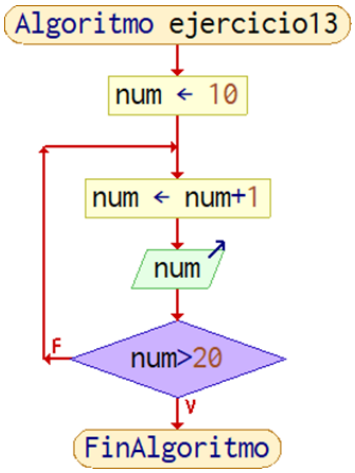


Figura 49: *Flujograma del algoritmo del uso de la estructura repetir segunda versión.*

Es importante resaltar que cuando se tratan de valores numéricos, el uso de combinaciones u operaciones aritméticas permiten la aplicación de artificios matemáticos que hacen posible una nueva alternativa de solución y con ello el resultado visiblemente, desde la pantalla (lo que ve el usuario), siga siendo el mismo. Te lo demostraremos en el siguiente ejercicio.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea presentar los números del 11 al 21 de forma descendente, (versión 3).

El algoritmo de esta versión 3, es el siguiente:

Algoritmo 14: Ejercicio14 -Números descendentes versión 3

num $\leftarrow$ 11
Repetir
    num $\leftarrow$ *num* $+1$
    Escribir (*num*- 1)
Hasta Que (num $>= 21 + 1$)

Fin Algoritmo

En el algoritmo anterior puedes notar que con un poco de ingenio matemático puedes usar valores que son reajustables por medio de una operación aritmética. La ubicación de la resta de menos 1 cuando se va a presentar el valor de la variable num hace que se muestre el valor sin el incremento, aun cuando este se haya efectuado.

```
1 Algoritmo ejercicio14
2     num ← 11
3     Repetir
4         num ← num + 1
5         Escribir (num - 1)
6     Hasta Que (num ≥ 21 + 1)
7 FinAlgoritmo
```

Figura 50: *Pseudocódigo del uso de la estructura repetir tercera versión.*

De igual manera en la condición, tener un 21 + 1 es igual a tener un 22 y con ello poder salir del lazo repetitivo mostrando el valor de hasta el 21, (ver Fig. 50). Si has notado, la estructura repetir permite siempre una primera ejecución del bloque que contiene, en este caso de las líneas 4 y 5 (pueden haber más). Y nuevamente se presenta en pantalla la secuencia de números del 11 al 21.

Estructuras repetitivas para

La estructura repetitiva para o lazo para, ejecuta el bloque de líneas cuando sabemos o conocemos el número de repeticiones que debamos realizar. Es decir, si se debe repetir o ejecutar 10 veces, el lazo irá de 1 al 10, veces de repeticiones.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea presentar los números del 10 al 20 de forma ascendente.

El algoritmo es el siguiente:

Algoritmo 15: Ejercicio15 -Números ascendentes del 10 al 20

Para $i = 10$ Hasta 20 Con Paso 1 Hacer
    Escribir i
Fin Para

Fin Algoritmo

En el algoritmo anterior se puede ahora observar que la estructura repetitiva para, tiene una sintaxis más reducida. Debido a que en una sola línea se encuentra la variable de inicialización, la condición de forma implícita y el incremento (ver Fig. 51).

```
1 Algoritmo ejercicio15
2     Para i = 10 Hasta 20 Con Paso 1 Hacer
3         Escribir i
4     Fin Para
5 FinAlgoritmo
```

Figura 51: *Pseudocódigo del uso de la estructura repetir segunda versión.*

En la figura 51, en la línea 2, el valor de i es igual a 10, la palabra “Hasta” representa la condición de llegar a 20 (incluido, en otros lenguajes no se lo incluye y ese será un tema para debatir, a observar y recordar) y las palabras “Con Paso” representarán el incremento de más 1 en la siguiente iteración.

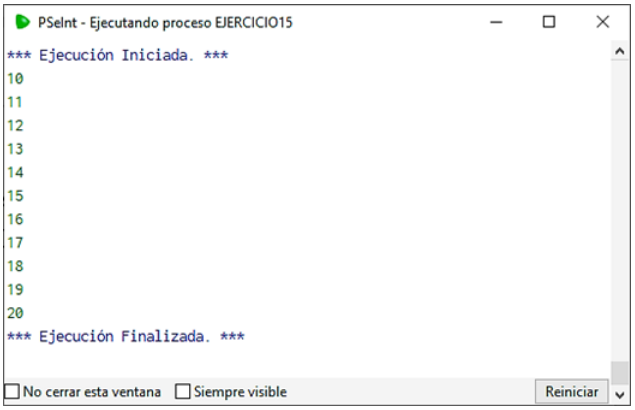


Figura 52: *Mensaje de salida del algoritmo con el uso de la estructura para.*

En la figura 52, se muestra la salida del algoritmo tal como indica el problema, mostrando los números del 10 al 20. Es necesario que la estructura no se reduce, es la misma que la estructura mientras o repetir, es su sintaxis que tiene una forma diferente.

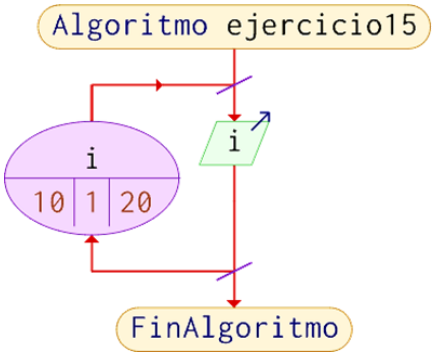


Figura 53: *Flujograma del algoritmo del uso de la estructura para.*

En el diagrama de flujo es visible notar la simbología circular que presenta, (ver Fig. 53). La variable de iteración i y el valor inicial de 10 como el valor final de 20 (inclusive). En el centro de los dos, se muestra el incremento. Debes observar la dirección de la flecha del lazo que indica el lazo, bucle, iteración o repetición de las sentencias dentro de ese segmento de la imagen. En un segundo ejemplo notaremos un cambio en el diagrama de flujo.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea presentar los números del 10 al 20 de forma descendente en pasos de 2.

El algoritmo es el siguiente:

Algoritmo 16: Ejercicio16 -Números descendentes del 10 al 20

Para $i = 20$ Hasta 10 Con Paso -2 Hacer
    Escribir i
Fin Para

Fin Algoritmo

Para este ejemplo de algoritmo, la variable de inicialización empieza en 20 y finaliza en 10 (inclusive), con ello se puede notar que los limites definen de dónde a dónde será el recorrido o repetición. El paso de menos dos indica como irá de forma descendente en pasos de -2.

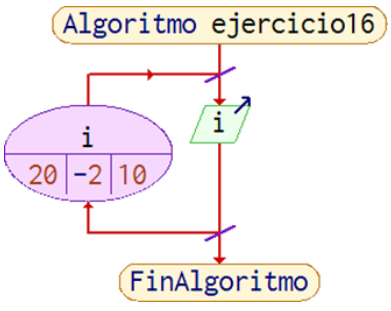


Figura 54: *Flujograma del algoritmo del uso de la estructura para.*

El diagrama de flujo (ver Fig. 54) presenta la simbología de un círculo donde se muestra la variable de iteración i, el valor de 20 es el primer valor inicial, el valor de 10 es el valor final, y el valor de -2 es el decremento que se realiza. Esto muestra que ahora el incremento de menos dos o el decremento es de dos, tal como se quiera leer y a la vez interpretar.

Funciones en PseInt

Una función es un bloque de código útil para ser usado muchas veces. Es necesario comprender es un subprograma que anteriormente ha sido ejecutado y probado. Es decir, es una subrutina o bloque de sentencias que se invocan cada vez que se las llama. Como aprendimos en matemáticas, las funciones polinómicas o trigonométricas, las funciones realizan procesos o subprocesos que en el programa principal se las usa.

Nos encontramos con dos tipos de funciones, aquellas que retornan un resultado o valor de un subproceso y aquellas que realizan algunas sentencias, que pueden mostrar resultados, pero no devolverlos o asignarlos a alguna variable. Vamos a revisar en el siguiente ejemplo una combinación de las dos.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea sumar cuatro parejas de números y luego presentar los números del 0 al 4. Dichas operaciones utilizando funciones.

El algoritmo es el siguiente:

Algoritmo 17: Ejercicio17 -Funciones

```
Funcion MostrarNumero
        Para i ← 0 Hasta 4 Con Paso 1 Hacer
                Escribir i;
        FinPara
Fin Funcion

Funcion result ← Sumar ( number1, number2 )
        definir result Como Entero;
        result ← number1+number2;
Fin Funcion

Algoritmo Algoritmo17 Funciones
        Escribir Sumar(3,5);
        Escribir Sumar(2,7);
        Escribir Sumar(4,1);
        Escribir Sumar(2,2);
        Escribir "*****"
FinAlgoritmo
```

Fin Algoritmo

Como puedes observar en este ejemplo en particular, tenemos varios bloques previos al algoritmo principal eso denota que debemos declarar, escribir, probar y estar seguros del código que usaremos en las funciones (ver Fig. 55).

```
1  Funcion MostrarNumero
2      Definir i como entero;
3      Para i←0 Hasta 4 Con Paso 1 Hacer
4          Escribir i;
5      FinPara
6  Fin Funcion
7
8  Funcion result ← Sumar ( number1,number2 )
9      definir result Como Entero;
10     result ← number1+number2;
11 Fin Funcion
12
13 Algoritmo Algoritmo17Funciones
14     Escribir Sumar(3,5);
15     Escribir Sumar(2,7);
16     Escribir Sumar(4,1);
17     Escribir Sumar(2,2);
18     Escribir "*****"
19     MostrarNumero()
20 FinAlgoritmo
```

Figura 55: *Pseudocódigo del uso de la estructura repetir segunda versión.*

De la línea 1 a la línea 6 tenemos nuestra función que lleva por nombre MostrarNumero, esta función no retorna nada o no devuelve nada, pero sí presenta algo (que no siempre es necesario). Utiliza una estructura para dentro de la función que va a permitir presentar los números de 0 al 4.

De la línea 8 a la línea 11, tenemos la segunda función del problema, lleva por nombre Sumar, tiene como parámetros dos variables, *number1* y *number2*. Retorna el resultado de lo que internamente realice a la variable *result* (ver Fig. 55). Esta función en cambio realiza un proceso de sumar dos números, *number1* y *number2* asignando el resultado a *result*. Podemos observar en las dos funciones que estamos revisando que es útil definir las variables con su tipo, como podemos observar en la línea 2 y en la línea 9, que son variables tipo entero.

De la línea 13 a la línea 20, encontramos el algoritmo o programa principal. La forma de llamar o invocar a cada función es diferente, para las funciones que retornan algo (asignan algo a una variable) encontramos su sintaxis en las líneas 14 a 17. En cambio, en la línea 18 se llama o invoca a la función que no retornan nada y se colocar su nombre junto a los paréntesis como está en la línea 19.
Al ejecutar el algoritmo para obtener su diagrama de flujo, notamos que se abre una ventana con la agrupación de todos los diagramas, es decir, PseInt presenta por bloques de gráficas cada bloque de pseudocódigo.

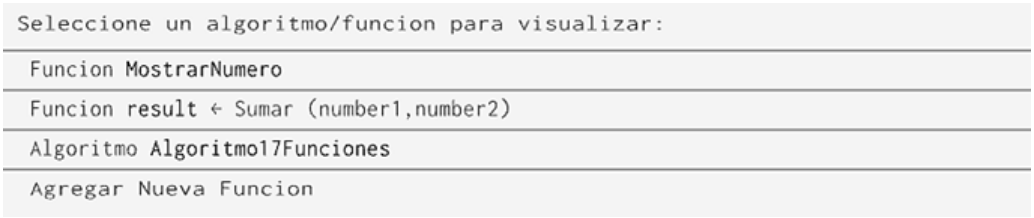


Figura 56: *Diagrama de flujo de agrupación de gráficas del algoritmo uso de funciones.*

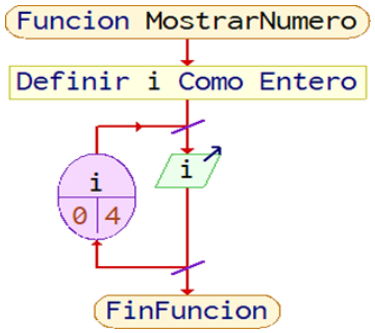


Figura 57: *Flujograma del bloque del algoritmo, función MostrarNumero.*

En la figura 57, podemos observar cómo se presenta la función *MostrarNumero* en un diagrama de flujo. Se muestra como un pequeño programa contenido dentro de este algoritmo. Por eso es necesario entender que se debe realizar primeramente el problema en pequeñas partes para luego unirlas.

Arreglos en PseInt

Los arreglos son un conjunto de datos del mismo tipo (y en algunos lenguajes son diferentes) uno a continuación de otro, llevan el mismo identificador (mismo nombre) y son referenciados por un valor llamado índice. Los arreglos sirven para almacenar los datos y de forma masiva. Recuerda que hasta ahora hemos trabajo con variables, pero imagina tener un gran número de ellas, será muy difícil recordar o darle un nombre a cada una. Por eso el uso de arreglos va a permitir tener muchos datos (como en Excel) y manipularlos con el nombre y un número que corresponde a su índice. A los arreglos se los conoce como vectores, debido a que nos debemos imaginar una fila de elementos uno a continuación de otros. Es como tener una línea de un punto inicial a un punto final, como un vector. Cuando estos valores se lo van tratando mediante unas operaciones como el ordenar, reubicar o insertar, llevan como nombre de lista. En algunos lenguajes estas estructuras pueden ser de tamaños fijos o ajustables, depende de lo que te permita el lenguaje. Es debido a que estos espacios son espacios de memoria y antes no se los podía desperdiciar, ahora con los nuevos lenguajes estos espacios pueden ser a medida mediante buenas prácticas de programación.

Ahora piensa en grande con más datos, e imagina que ya no tienes un vector sino muchos vectores que se van colocando de arriba hacia abajo, entonces puedes imaginar una matriz. Es decir, un conjunto de elementos colocados en filas y columnas, como un edificio, con pisos y en cada piso, varios departamentos.

Para terminar esta unidad vamos a observar varios ejercicios con arreglos y matrices.

Escriba un algoritmo en PSeInt y su representación en su diagrama de flujo para resolver el siguiente problema: Se desea almacenar un conjunto de nombres de personas y sus edades. Utilice un arreglo para el almacenamiento. Para operaciones adicionar y listar (presentar lo almacenado) utilice funciones. Presente un menú de opciones para el usuario.

El algoritmo es el siguiente:

Algoritmo 18: Ejercicio18 -Arreglos

```
Funcion Listar(n Por Referencia,e Por Referencia,p Por Referencia)
    definir index Como Entero;
    Para index← 1 Hasta p − 1 Con Paso 1 Hacer
        Escribir n[index],“: ”,e[index];
    FinPara
Fin Funcion

Funcion add(n Por Referencia,e Por Referencia,p Por Referencia)
    Escribir Ïngrese Nombre";
    Leer n[p];
    Escribir Ïngrese Edad";
    Leer e[p];
```

```
        p←p+1;
    Fin Funcion

    Funcion Menu()
        Definir op, puntero,edades como entero;
        Definir nombres como caracter;
        Dimension nombres[100];
        Dimension edades[100];
        puntero← 1;
        repetir
            Escribir "1. Ingresar";
            Escribir "2. Listar";
            Escribir "0. Salir";
            Leer op;
            Segun op hacer
                1:
                    add(nombres, edades, puntero);
                2:
                    Listar(nombres, edades, puntero);
            FinSegun
        Hasta Que op=0;
    Fin Funcion

    Algoritmo Algoritmo16 Arreglo
        Menu();
    FinAlgoritmo
```

```
Fin Algoritmo
```

Se puede observar que este algoritmo es más extenso, es decir, mientras vayamos añadiendo operaciones a realizar, deberemos usar más código para lograr lo requerido. Por eso debes ser ordenado y analizar bien el problema para dividirlo por partes y encontrar la solución. Vamos analizando por líneas.

```
1  Funcion Listar(n Por Referencia,e Por Referencia,p Por Referencia)
2      definir index Como Entero;
3      Para index←1 Hasta p-1 Con Paso 1 Hacer
4          Escribir n[index],": ",e[index];
5      FinPara
6  Fin Funcion
```

Figura 58: *Pseudocódigo del uso de arreglos y funciones, función Listar.*

Si observas las primeras líneas de la 1 a la 6 tenemos la función *Listar*, esta función tiene tres parámetros que le damos desde el algoritmo principal o desde quien la invoque. Es como tener tan(x), tangente de x, debemos pasarle x a esa función trigonométrica. Lo mismo sucede con la

función *Listar*, debemos pasarle o entregarle valores en ese orden, valor de *n*, valor de *e* y valor de *p* respectivamente. Por eso debes observar que tiene un texto en azul "Por Referencia" entre paréntesis, (ver Fig. 58).

En la línea 2 se tiene una variable index definida como "Entero". Desde la línea 3 a la 5 tenemos una estructura lazo Para, donde empiezan las repeticiones desde el valor de *index* con 1 hasta el valor de la variable *p* menos 1, en pasos de 1. Lo que está dentro de la estructura *Para*, es lo que repetirá *index* veces. El contenido del arreglo *n* en la posición *index* separado de dos puntos y luego el contenido del arreglo *e* en la posición *index*. Finalmente se cierra la función. Debes observar más adelante quién la llama o la invoca y qué valores le pasa.

```
8  Funcion add(n Por Referencia,e Por Referencia,p Por Referencia)
9      Escribir "Ingrese Nombre";
10     Leer n[p];
11     Escribir "Ingrese Edad";
12     Leer e[p];
13     p←p+1;
14 Fin Funcion
```

Figura 59: *Pseudocódigo del uso de arreglos y funciones, función add.*

De la línea 8 a la línea 14 encontramos la función llamada *add*, también notamos que entre paréntesis se encuentran tres variables como parámetros, estas variables son *n*, *e* y *p*. Como vimos en la función anterior, *n* es un arreglo, *e* es un arreglo y *p* una variable que incrementa. A simple vista, esta función *add*, pide al usuario que ingres el nombre, y se almacena en el arreglo *n* en la posición *p*. Lo mismo hace en las líneas 11 y 12, donde pide al usuario que ingrese la edad y luego almacena lo ingresado en el arreglo *e* posición *p*. La línea 13 es importante entenderla, incrementa a *p* para la siguiente vez que se invoque a la función, es decir deja lista la siguiente posición a llenar en los dos arreglos gracias a ésta última sentencia, (ver Fig. 59).

En las líneas 16 a la línea 34, encontramos a la función *Menu*, esta función define a las variables *op*, puntero y edades como entero, y a nombres como carácter, es decir usará cuatro variables mientras se la invoque. Además, tiene declarados dos arreglos, nombres y edades cada uno de tamaño o dimensión 100. En la línea 21 encontramos la variable que hace de puntero o apuntador, asignando el valor de 1, (ver Fig. 60).

De la línea 22 a la línea 33 encontramos la estructura repetitiva llamada repetir, y que la controla la variable *op*, hasta que ésta sea cero. Esta estructura repetir, es muy usada para la creación de menús, donde las primeras líneas son mensajes como se muestra en las líneas 23, 24 y 25. La línea 26, es la variable op, que lee lo que ingresa el usuario, es decir de acuerdo con ese ingreso se repite o se sigue a otra sentencia selectiva que está más abajo, *según*.

```
16  Funcion Menu()
17      Definir op, puntero,edades como entero;
18      Definir nombres como caracter;
19      Dimension nombres[100];
20      Dimension edades[100];
21      puntero←1;
22      repetir
23          Escribir "1. Ingresar";
24          Escribir "2. Listar";
25          Escribir "0. Salir";
26          Leer op;
27          Segun op hacer
28              1:
29                  add(nombres, edades, puntero);
30              2:
31                  Listar(nombres, edades, puntero);
32          FinSegun
33      Hasta Que op=0;
34  Fin Funcion
```

Figura 60: *Pseudocódigo del uso de arreglos y funciones, función Menu.*

La estructura según valida a *op* en la línea 27, y de acuerdo con lo que el usuario entiende por los mensajes (1, 2 o 0) el algoritmo invocará en la opción 1 a la función *add*, en la opción 2, invocará a la función *Listar*. Si observas en la línea 29, la función *add* es llamada (invocada), y se le pasa o agrega el valor de nombres, edades y del puntero. Si observas en la línea 31, la función *Listar* es llamada (invocada), y se le pasa o agrega el valor de nombres, edades y del puntero.

```
36  Algoritmo ejercicio18
37      Menu();
38  FinAlgoritmo
```

Figura 61: *Pseudocódigo del algoritmo uso de arreglos y funciones.*

En la línea 36 está el algoritmo y su nombre, en la línea 37 se invoca a la función Menu(), recuerda que algoritmos y lenguajes de programación, las tildes o acentos no se usan por eso *Menu* no lleva tilde (ver Fig. 61).

Puedes observar que si unimos los bloques funciona el programa y algo muy interesante es que lo has divido en partes para su mejor comprensión. Esta práctica proviene de la idea de dividir el problema y vencerás. Además, puedes notar que si sólo viéramos el algoritmo sería muy corto o pequeño y de él se llama a una primera función y de esta a otras dos. Son buenas prácticas que debes analizar de otros códigos y valorar su uso.

En la figura 62, se observa la salida del algoritmo cuando lo ejecutamos o lo hacemos funcionar. Se ve en la ventana que se ingresa el nombre y luego la edad en la opción 1 escogida, con ello si vamos a la opción 2, se podrá listar lo ingresado. Si validamos un poco más los posibles escenarios pudiéramos enviar mensajes en la opción 2 si no hay nada que listar (mostrar).

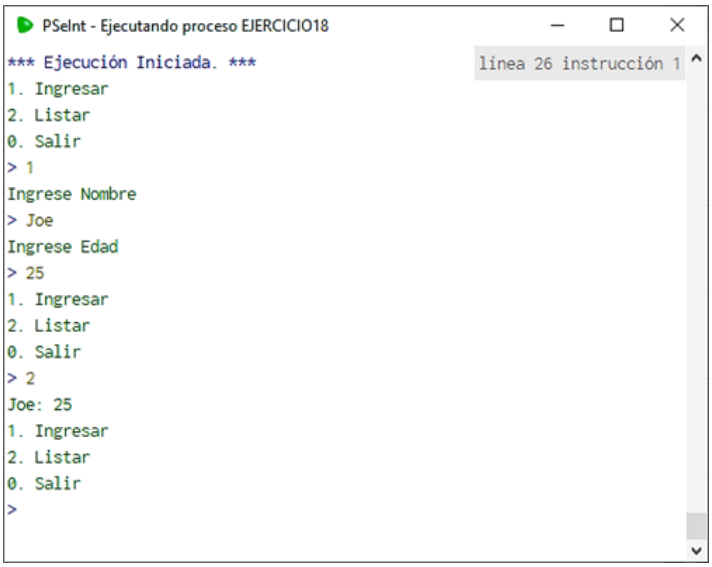


Figura 62: *Pseudocódigo del algoritmo uso de arreglos y funciones.*

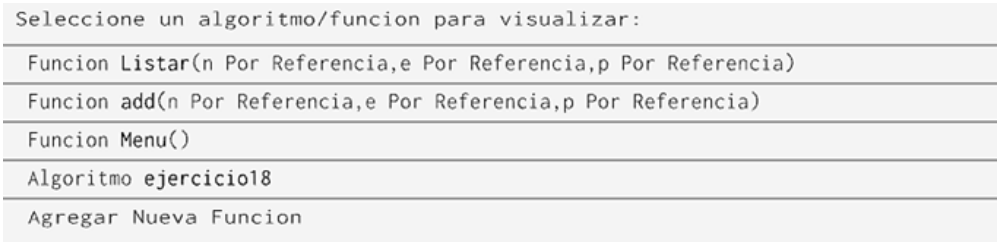


Figura 63: *Pseudocódigo del algoritmo uso de arreglos y funciones.*

Finalmente, si deseamos observar las gráficas de los diagramas de flujo, se mostrará una ventana para escoger cuál diagrama deseas ver. Recuerda aprender funciones es como crear pequeños programas que al funcionar se pueden reutilizar. Y al estudiar arreglos o matrices puedes con ellos almacenar grandes cantidades de datos o valores.

2.3 Experiencia concreta [Sentir] -Actuar

"*Que el contenido de esta unidad sea un desafío de representar algoritmos mediante una adecuada solución*". Tus profesores.

2.3.1 DESAFÍO

Vamos a realizar un ejercicio donde te piden la multiplicación por el método ruso usando variables y luego usando arreglos con funciones. Este método indica que dos números se multiplican así, el primero se lo divide dos para tantas veces hasta llegar a uno, de igual manera al segundo se lo multiplique por dos mientras el primero llegue a uno. Si al que se divide su resultado entero es impar se suma su correspondiente número (que se multiplica). La suma de estos números debe dar el mismo resultado que si se hubiere multiplicado los primeros números.

Desafío 4. Elabore un algoritmo que aplique las estructuras repetitivas

Representar el método de multiplicación rusa utilizando variables

El algoritmo, es el siguiente:

Algoritmo 19: Ejercicio19 -Ruso1

```
Escribir Ïngrese un primer número: "
Leer n1
Escribir Ïngrese un segundo número: "
Leer n2
nu1 = n1
nu2 = n2
acum = 0
Repetir
        Escribir n1,, n2
        Si (n2 Mod 2 == 1) Entonces
                acum = acum + n1
        Fin Si
        n1 = n1 * 2
        n2 = trunc(n2 / 2)
Hasta Que n2 <= 0
Escribir nu1,“ por ”, nu2,“ es ”, acum
Escribir "que es igual a ", (nu1*nu2)
```

Fin Algoritmo

Si revisas el algoritmo, se solicita al usuario dos números y luego se les saca una copia.

La línea 6 y 7 permiten tener un duplicado de los dos primeros números. Esto se realiza debido a que los valores originales se perderán durante el proceso. Es decir, n1 y n2 se los tratarán durante las siguientes líneas que sus valores originales se perderán (ver Fig. 64).

```
1  Algoritmo ruso1
2      Escribir "Ingrese un primer número: "
3      Leer n1
4      Escribir "Ingrese un segundo número: "
5      Leer n2
6      nu1 = n1
7      nu2 = n2
8      acum = 0
9      Repetir
10         Escribir n1," ",n2
11         Si (n2 Mod 2 == 1) Entonces
12             acum = acum + n1
13         Fin Si
14         n1 = n1 * 2
15         n2 = trunc(n2 / 2)
16     Hasta Que n2 ≤ 0
17     Escribir nu1," por ",nu2," es ", acum
18     Escribir "que es igual a ",(nu1*nu2)
19 FinAlgoritmo
```

Figura 64: *Pseudocódigo del algoritmo del método ruso con variables.*

Desde la línea 9 hasta la línea 16 encontramos la estructura repetitiva que realiza el proceso de multiplicar al primer número y de dividir al segundo número (líneas 14 y 15). La estructura repetitiva *repetir* aquí es útil debido a que no sabemos cuántas veces se producirán estos dos procesos. Y es el proceso de división quién controla al lazo, debido a que las divisiones se dan hasta que se llegue a un valor menor a 1 (el cero). Finalmente, el acumulador guardará al primer número si el segundo, su división entera sea impar.

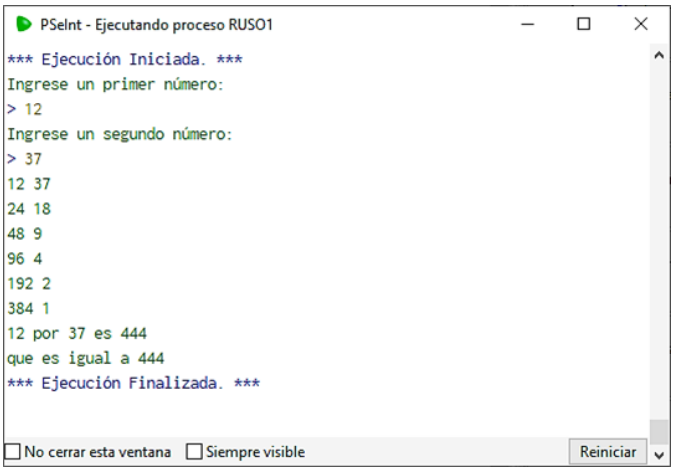


Figura 65: *Pseudocódigo del algoritmo del método ruso con variables.*

Este método de multiplicación tiene una razón de hacer procesos de forma en conjunta, por eso, al usar variables debemos guardar un duplicado. La utilidad del sí-entonces validará un valor mientras se repitan los procesos. Te desafiamos a realizar un dibujo de cómo se realizan los procesos, (ver Fig. 66).

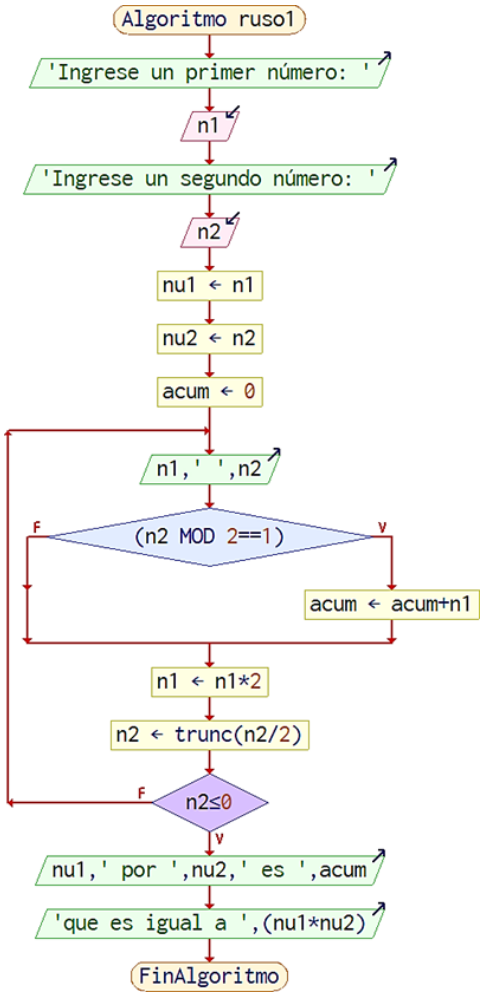


Figura 66: *Flujograma del algoritmo del método ruso con variables.*

Desafío 5. Elabore un algoritmo que aplique el uso de los arreglos

Representar el método de multiplicación rusa utilizando arreglos

El algoritmo, es el siguiente:

Algoritmo 20: Ejercicio20 -Ruso2

```
Algoritmo ruso2
Dimension arre1[100];
Dimension arre2[100];
i = 1
Escribir 'Ingrese un primer número: '
Leer arre1[i]
Escribir 'Ingrese un segundo número: '
Leer arre2[i]
acum ← 0
Repetir
        Escribir arre1[i],' ', arre2[i]
```

```
        Si (arre2[i] MOD 2==1) Entonces
            acum ← acum + arre1[i]
        FinSi
        arre1[i+1] ← arre1[i]*2
        arre2[i+1] ← trunc(arre2[i] / 2)
        i = i + 1
    Hasta Que arre2[i] <= 0
    Escribir arre1[1],' por ', arre2[1],' es ', acum
    Escribir 'que es igual a ', (arre1[1]*arre2[1])
Fin Algoritmo
```

Si revisas el algoritmo, se utilizan arreglos para este problema. Da la apreciación que tiene la misma lógica de solución al problema anterior, pero ahora no hay variables n1 y n2, ni tampoco un duplicado, (ver Fig. 67).

```
1  Algoritmo ruso2
2      Dimension arre1[100];
3      Dimension arre2[100];
4      i = 1
5      Escribir 'Ingrese un primer número: '
6      Leer arre1[i]
7      Escribir 'Ingrese un segundo número: '
8      Leer arre2[i]
9      acum ← 0
10     Repetir
11         Escribir arre1[i],' ',arre2[i]
12         Si (arre2[i] MOD 2==1) Entonces
13             acum ← acum + arre1[i]
14         FinSi
15         arre1[i+1] ← arre1[i]*2
16         arre2[i+1] ← trunc(arre2[i] / 2)
17         i = i + 1
18     Hasta Que arre2[i] ≤ 0
19     Escribir arre1[1],' por ',arre2[1],' es ',acum
20     Escribir 'que es igual a ',(arre1[1]*arre2[1])
21 FinAlgoritmo
```

Figura 67: *Flujograma del algoritmo del método ruso con variables.*

Lo que llama la atención es que al pensar y usar arreglos son sus posiciones que guardan valores y luego estos pueden ser utilizados.

2.4 Reflexión [Analizar] - Observar

"*Que el contenido de esta unidad sea un desafío de representar algoritmos mediante una adecuada solución*". Tus profesores.

2.4.1 EVALUACIÓN

[Lea la página 4 y siga el formato para la entrega de todas las actividades, en un solo documento]

Validación del aprendizaje

Validación del aprendizaje de la Unidad 2

- Elabora tres funciones que permitan encontrar (a) la potencia de un número como 2^4 y que presente el resultado esperado, para este ejemplo, 16, (b) el factorial de un número, por ejemplo, $4!$, que sería las multiplicaciones de $1 \times 2 \times 3 \times 4$ dando como resultado 24 y (c) la secuencia de el enésimo término de la serie de Fibonacci, por ejemplo fibo(7) y presente los 7 primeros números de la seria, que son 0 1 1 2 3 5 8 .
- Elabora un algoritmo que permita crear 5 arreglos de 50 elementos y llenar de ceros las posiciones pares y de unos las posiciones impares.

Reactivos de la Unidad 2

Pregunta 1. Dado el siguiente algoritmo, determine el valor de las variables al finalizar su ejecución.

Algoritmo ejercicio	**a**	**b**	**c**
a=2			
b=a-1			
c=(a*b)-2			
b=a+(3+b)			
a=b-2-c			
FinAlgoritmo			

a = 4, b = 4, c = 0	
a = 6, b = 6, c = 0	
a = 4, b = 6, c = 0	
a = 0, b = - 6, c = 0	

Pregunta 2. Dado el siguiente Algoritmo, determine la respuesta correcta luego de evaluar lo solicitado.

```
Algoritmo examen
    Repetir
        Escribir "1. Programa 1"
        Escribir "2. Programa 2"
        Escribir "3. Programa 3"
        Escribir "5. Salir"
        Leer op
        Segun op Hacer
            1:  Leer n1
                Si n1 MOD 2 == 0 Entonces
                    Escribir n1, " primer mensaje"
                SiNo
                    Escribir n1, " segundo mensaje"
                Fin Si
            2:  cont = 0
                Leer n2
                Para i=1 Hasta n2 Con Paso 1 Hacer
                    re = n2 MOD i
                    Si re == 0 Entonces
                        cont = cont + 1
                    Fin Si
                Fin Para
                Si cont <= 2 Entonces
                    Escribir n2," tercer mensaje"
                SiNo
                    Escribir n2," cuarto mensaje"
                Fin Si
            3:  fac = 1
                Leer n5
                Para i = 1 Hasta n5 Con Paso 1 Hacer
                    fac = fac * i
                Fin Para
                Escribir "quinto mensaje: ", fac
            5:  Escribir "Salimos del programa"
            De Otro Modo:
                Escribir "Opción incorrecta"
        Fin Segun
    Hasta Que op == 5
FinAlgoritmo
```

Si se ingresa al algoritmo, el número 2 y luego el número 9, se obtiene como salida.

7 quinto mensaje	
8 tercer mensaje	
9 cuarto mensaje	
mensaje incorrecto	
Salimos del programa	

Unidad 3

% Unidad 3

Capítulo 3 Desarrollo de la Unidad 3

"*Que el contenido de esta unidad sea el reto final de crear programas que solucionen problemas reales*". Tus profesores.

Unidad 3: Lenguajes de Programación

a) **Tiempo para su desarrollo:**

Total semanas:	8	**Total de horas:**	32
Fecha de inicio:		**Fecha finalización:**	
Componente Docencia	Componente Práctica de Aplicación y Experimentación	Componente Trabajo Autónomo	Total
32	16	32	80

b) **Presentación de la unidad**

Breve descripción de la unidad y los resultados de aprendizaje esperados.

c) **Actividades de aprendizaje estudiantil**

- Lectura, estudio, análisis de los contenidos relacionados a la unidad (bibliografía adjunta).
- Aplicación concreta de los contenidos conceptuales trabajados a través de la lectura.

d) **Recursos para las actividades de aprendizaje estudiantil**

- Texto 1 que consta al final de esta guía y/o archivo digital.
- Aplicación concreta de los contenidos conceptuales trabajados a través de la lectura.

e) **Cronograma sugerido**

Actividad académica	**Fecha de inicio**	**Fecha de finalización**	
Lectura de la temática			10 horas
Generación del trabajo académico a entregarse (Portafolio Estudiantil)			22 horas
Fecha de entrega al docente			

f) **Criterios para la evaluación de los trabajos académicos Unidad 3**

- Autoevaluación: Cada estudiante valora su actividad académica, de acuerdo con el formato establecido, tributa a ser justo y honesto con uno mismo.
- Evaluación: El docente valora el trabajo académico de cada estudiante, de acuerdo con el formato específico elaborado para el efecto. Tributa a dar ejemplo concreto a sus estudiantes, de los valores que debe practicar todo buen docente.
- Evaluación integral: Contribuye a una visión más cercana y equilibrada de la realidad evaluativa, contribuye a una evaluación formativa, que no busca sancionar sino, ante todo mejorar la realidad estudiantil involucrando valores y exigencia académica y de

esta manera contribuir a la Misión universitaria: formar honrados ciudadanos y buenos cristianos, con capacidad académica e investigativa que contribuyan al desarrollo sostenible local y nacional.

- De acuerdo con el artículo 41 del Reglamento de Régimen Académico de la Universidad Politécnica Salesiana, la evaluación tiene la característica de ser sistemática y continua, misma que considera un total de cien (100) puntos.

g) **Rúbrica de evaluación**

CRITERIO	DESCRIPCIÓN DEL CRITERIO	Excelente 80 %	Muy bueno 70 %	Bueno 60 %	Regular 40 %	Por mejorar menos de 40 %
Responsabilidad Académica	Experiencia concreta [Sentir] – Actuar					
	Reflexión [Analizar] - Observar					
Las actividades académicas las he realizado con:						
Responsabilidad formativa	Honestidad académica: sin plagio o copia	10 %	20 %	20 %	20 %	20 %
	Dedicación responsable	5 %	5 %	5 %	5 %	5 %
	Puntualidad en la entrega	5 %	5 %	5 %	5 %	5 %
TOTAL sobre 100 % del puntaje asignado						

Observaciones y retroalimentación	
Firma y nombre de:	
Estudiante	Profesor

3.1 Conceptualización [Profundizar] - Pensar

"*Que el contenido de esta unidad sea el reto final de crear programas que solucionen problemas reales*". Tus profesores.

3.1.1 LENGUAJES DE PROGRAMACIÓN

Los lenguajes de programación son aquellos programas que permiten crear nuevos programas. Su función es de comunicarse con el computador a través de un lenguaje que entienda la máquina, siendo este compilado o interpretado. El tipo de lenguaje utilizado contendrá una manera de escritura que se conoce como sintaxis y que los programadores con la práctica y el estudio podrán realizar algoritmos y sus variantes que se transforman en programas de computadora [**41**, **42**, **43**, **44**, **45**, **46**].

Existen muchos lenguajes en la actualidad, y el que se usa en esta guía será Python, de la familia de versiones 3. Motivamos al uso de MATLAB como un segundo lenguaje para cálculos especialmente para el tratamiento de matrices [**47**]. En el enlace del índice TIOBE puedes tener una idea histórica de aquellos lenguajes de programación usados con más frecuencia en un año determinado[6].

Así como todos los días realizamos actividades rutinales, los algoritmos son aquellos pasos para desarrollar una actividad, las computadoras utilizan programas para ejecutar sentencias que con ellas define el desarrollo de una determinada actividad. Podemos observar programas sencillos como en una máquina expendedora de botellas, el programa que funciona en el cajero automático o aquel programa que usamos en una aplicación móvil, todos ellos utilizan algoritmos convertidos en programas y en muchos de ellos se reutilizan algunos de sus funciones (código) en nuevas versiones.

Recuerda un poco cómo ha ido cambiando Microsoft Word y sus nuevas versiones que nos permiten entender las características fundamentales (de escritura) y muchas de ellas permanecen en el tiempo y a las nuevas versiones se les van agregando otras funcionalidades. Así y en esa forma se debe aprender a programar. Realizar prototipos funcionales (con eficiencia en su desarrollo) y que luego reescribirlos para mejorarlos.

A esto se lo conoce como las fases de desarrollo de un programa (ver Fig. 68), que si se repiten podemos llamarlo un ciclo.

Figura 68: *Ciclos de desarrollo de un programa.*

[6]Disponible en https://www.tiobe.com/tiobe-index/

Python fue creado a finales de 1989 por Guido van Rossum y él lo describió como: fácil, intuitivo, de código abierto y potente [**48**, **49**, **50**]. Comprensible como el texto en versión de lenguaje en inglés [**51**]. Su utilidad, tanto para hacer sencillos programas permite también ser usado para programar en áreas como la inteligencia artificial, analíticas de los datos, como para dispositivos en internet de las cosas entre otros campos [**52**, **53**, **54**, **55**].

De la misma manera, MATLAB, o *laboratorio matricial*, es un programa matemático que fue creado en 1984, con un entorno de desarrollo (IDE), por Cleve Moler en Mathworks, con un lenguaje propio llamado lenguaje M. Su utilidad se dirige a dar solución a problemas de análisis numérico enfocado a las matrices y manejo de grandes cantidades de cálculos de datos [**56**, **57**, **58**]. Al igual que Python, MATLAB utiliza una sintaxis muy similar y ellos logran ser en la actualidad aquellas alternativas a las necesidades computaciones de matemáticos, ingenieros y científicos.

El desafío de aprender estos dos lenguajes y sus entornos serán los objetivos que iremos descubriendo a continuación.

3.2 Experiencia activa [Experimentar] - Hacer

"*Que el contenido de esta unidad sea el reto final de crear programas que solucionen problemas reales*". Tus profesores.

3.2.1 LENGUAJE PYTHON

Para el desarrollo de programas en lenguaje Python, debemos instalar desde `https://www.python.org/` la última versión disponible. Alternativas para el desarrollo puede ser `https://anaconda.org/`, también PyCharm que puedes descargar de `https://www.jetbrains.com/es-es/pycharm/download/#section=windows` o tu laboratorio en línea sobre la plataforma de Google (usando una cuenta de Gmail para tener acceso a la plataforma de Google Colaboratory o Google Colab) desde `https://colab.research.google.com/notebooks/intro.ipynb`, finalmente puedes acceder a un entorno desde tu dispositivo móvil con alguna aplicación para estudio, (ver Fig. 69).

Figura 69: *Ejemplo de Aplicaciones móviles disponibles para aprender Python.*

Es indispensable que llegues a instalar un entorno de desarrollo en tu computador para usar el lenguaje de programación adecuado. Con la ayuda entornos colaborativos en la web se puede también compartir el código y trabajar en grupos. Por ahora trabajaremos desde el equipo o dispositivo que se tiene. Puedes ayudarte con lápiz y papel.

Para los ejercicios que se muestran a continuación podemos utilizar plataformas gratuitas de desarrollo en línea (ver Fig. 37) así como instaladas en el equipo, y aún más, si no tuviéramos un entorno de desarrollo, con el simple lápiz y papel podremos escribir el código y seguir un proceso como prueba de escritorio que nos ayudará en la lógica del entender el programa.

La sintaxis dependerá de la buena observación en lo que escribimos y cómo usemos los signos o caracteres especiales, como los paréntesis, comas, comillas o, la identación, la separación

(tabulación) o alineación vertical desde la izquierda y siguiendo un orden, entre otros.

Figura 70: *Ejemplo de entorno web de Google Colaboratory*

Lo más fundamental de este inicio de unidad es que debes lograr entender el problema, y es ahí donde todo comienza. Entender el problema significa que debe recoger toda la información que te dan en el enunciado y entenderlo. Dividirlo en pequeñas secciones y tratar de relacionarlo con lo estudiado.

Uno de los primeros pasos es detectar las variables de ingreso (valores de ingreso) y variables de salida (valor a presentar). Luego determinar las estructuras selectivas, condicionales y repetitivas que utilizarás mientras avanzas en el entendimiento del problema. Finalmente escribir un primer prototipo, luego otro hasta que puedas optimizarlo. Dividir el problema podrá ayudarte a identificar si es posible convertir en función una sección del algoritmo de tal forma que dicha solución pueda ser utilizado nuevamente en otros problemas.

3.2.2 SALIDA Y ENTRADA DE DATOS

Salida o mensaje

Codifique un programa que presente en pantalla o a la salida un mensaje de "*hola a todos*". El código utilizaría una función (usa paréntesis) llamada `print()`.

Código 1: hola1.py

```
# Comentario: hola1.py es el nombre del archivo
print("Hola a todos")
```

El mensaje en pantalla sería:

```
Hola a todos
```

El contenido que va dentro de los paréntesis se lo conoce como argumento o argumentos. Estos argumentos pueden ser variables o palabras reservadas que realizan algo. En el ejemplo anterior lo que está entre comillas se lo conoce como un tipo de dato llamado string, que sería una cadena de caracteres. Y existen acciones y/o valores por defecto que se deben apreciar. Por ejemplo, un espacio vacío o un salto de línea.

Codifique un programa que presente en pantalla o a la salida un mensaje de "*hola a todos*" con saltos de línea.

Código 2: hola2.py

```
# Comentario: hola2.py es el nombre del archivo
print("Hola a todos")
print()
print("Hola ")
print("a ")
print("todos \n")
print("Hola a todos", end=' * ')
print("Hola a todos")
```

El mensaje de salida que se produce será:

```
Hola a todos
Hola
a
todos
Hola a todos * Hola a todos
```

Si analizamos línea a línea, la primera línea es similar al primer mensaje que es mostrado como en el código anterior. La segunda línea por defecto es un salto de línea que tiene *print()*. La tercera, cuarta y quinta línea es la presentación de cada palabra, note que se muestra luego de un salto de línea. Es decir, cada instrucción equivale a una línea en la salida. Si observa bien, en la quinta línea existe un carácter especial "\n" que representa un salto de línea. Por eso no requiere la sexta línea un *print()* vacío. La sexta y séptima línea, aunque cada una está en una línea (una instrucción), estas se encuentran unidas. La función *print()* tiene un argumento llamado *end*, que se le puede asignar un caracter o ninguno. Para este ejemplo se ha colocado el argumento *end* para que se muestren en una misma línea y, entre líneas un asterisco.

Otra forma de escribir nuestro programa es definiendo una función. Esta debe tener un nombre.

Código 3: hola3.py

```
# Comentario: hola3.py es el nombre del archivo
def hola():
      print("Hola a todos")
hola ()
```

El mensaje de salida sería:

```
Hola a todos
```

El lenguaje Python es sensible en su escritura, por ejemplo, la segunda línea visible o línea interna de la función hola, está desplazada o tiene una tabulación por defecto, esta tabulación es necesaria ya que dentro de una estructura (en este caso una función) el bloque de código debe estar identado (tabulado), asimismo más adelante se muestra que el lenguaje Python distingue las mayúsculas de las minúscula.

Expresiones LINEALES

Si se desea escribir una expresión matemática como, por ejemplo:

$$x = \frac{-b - (b^2 - 4ac)}{2a + \frac{1}{3}}$$

Se debe escribir todo en forma lineal (en una sola línea al escribir) y utilizar los paréntesis necesarios guardando la prioridad de los operadores aritméticos.

Código 4: expresion.py

```
# Comentario: expresion.py es el nombre del archivo
# Recuerda el uso parentético (entre paréntesis)
x = (-b-(b*b -4*a*c))/(2*a + (1/3))
```

Como puedes observar se requerirá usar valores variables y valores constantes, operadores aritméticos, y funciones matemáticas o trigonométricas.

Entrada de datos

Codifica un programa que permita encontrar el área de un círculo. Para ello, utilice el valor de la variable PI con un valor constante de 3.141516 aproximadamente. El valor del radio se lo ingresa desde teclado.

Código 5: area_circulo1.py

```
# Comentario: area_circulo1.py es el nombre del archivo
import math as mat
print("Ingrese el valor del radio: ", end=' ')
radio = int(input())
area = mat.pi * radio * radio
print("El resultado es: ", area)
```

El mensaje en pantalla sería:

```
Ingrese el valor del radio: 1
El resultado es: 3.141592653589793
```

Utilizando lo aprendido y ahora ingresando valores, explicaremos cada línea del código 5. La primera línea se importa el paquete matemático de la clase *math* que contiene fórmulas y valores constantes para usar. Se lo renombra como *mat* para hacer más comprensible el código en la cuarta línea. La segunda línea de código utiliza la función *print()* con un primer argumento de la cadena de caracteres entre comillas y un segundo argumento con *end* para que la tercera línea de ingreso del valor se muestre al final del mensaje *Ingrese el valor del radio:* _.

La tercera línea hay que entenderla en dos partes, recuerde que cuando vea paréntesis son funciones. Entonces *input()* es la función propia para ingreso de valores en Python. La función *int()* en cambio traduce lo que se ingresa por teclado transformando en un número entero, asignándolo

a la variable radio. La cuarta línea representa una expresión matemática de la fórmula del área del círculo. Se observa que existe un valor constante llamado pi que se lo invoca a través de *mat*. Asimismo, la operación del cuadrado de un número puede ser mejor expresado y de manera sencilla al multiplicar dos veces la variable radio. Finalmente, la última línea representa la información obtenida de la línea anterior y mostrada a través del print, separando la cadena de caracteres del mensaje y de la variable del resultado llamada *area*.

Código 6: area_circulo2.py

```
# Comentario: area_circulo2.py es el nombre del archivo
import math as mat
radio = int(input("Ingrese el valor del radio: "))
area = mat.pi * float(radio**2)
print("El resultado es: ", area)
```

El resultado en pantalla sería:

```
Ingrese el valor del radio: 1
El resultado es: 3.141592653589793
```

Si reformulamos el código que se ha aprendido, la segunda línea se ha unificado permitiendo quitar el *print()*, *input()* permite agregar un mensaje como argumento. En la tercera línea también hay dos cambios, la potencia de un número puede ser calculada con el uso de los dos asteriscos. Además, la función *float( )* permite realizar la operación entre números reales.

Codifique un programa que calcule el área del triángulo en función de las longitudes de sus lados, donde $p = (a + b + c)/2$ es el semiperímetro:

$$area = \sqrt{p(p-a)(p-b)(p-c)}$$

Código 7: area_tri.py

```
# Comentario: area_tri.py es el nombre del archivo
import math as mat
a = int(input("Ingrese el valor de a: "))
b = int(input("Ingrese el valor de b: "))
c = int(input("Ingrese el valor de c: "))
p = (a+b+c)/2.0
are = mat.sqrt(p*(p-a)*(p-b)*(p-c))
print("El resultado es: ", are)
```

El resultado en pantalla sería:

```
Ingrese el valor de a: 3
Ingrese el valor de b: 2
Ingrese el valor de c: 3
El resultado es: 2.8284271247461903
```

Del código 7, utilizamos lo aprendido, la línea primera, sirve para importar las librerías de la clase *math*, renombrando como mat, y utilizar en la sexta línea de código la función raíz cuadrada llamada *sqrt()*. Las líneas, segunda, tercera y cuarta permiten el ingreso de tres valores con sus respectivos mensajes al usuario. La quinta línea, se obtiene el semiperímetro, la suma de las tres variables, *a*, *b* y *c* divido para dos. Observe que al colocar 2.0 el resultado será un valor real. La sexta línea si se observa se invoca la función *sqrt()*, para obtener la raíz cuadrada. Finalmente, la séptima línea se muestra el resultado.

PRUEBA DE ESCRITORIO

Codifique un programa que almacene valores en las variables a, b y c, luego realice varias operaciones aritméticas asignando y sobrescribiendo sus valores.

Código 8: prueba_escritorio.py

```
# Comentario: prueba_escritorio.py es el nombre del archivo
a = int(input("Ingrese el valor de a: ")) # 2
b = int(input("Ingrese el valor de b: ")) # 5
c = a + b
c = a + 2
b = c + 1
a = b + 3
c = 0
print(a, " ", b, " ", c, " ")
```

El resultado en pantalla sería:

```
Ingrese el valor de a: 2
Ingrese el valor de b: 5
8   5   0
```

Para este ejercicio, debemos utilizar una tabla para desarrollar línea a línea el proceso de ejecución de instrucción por instrucción. A esto se lo conoce como prueba de escritorio.

Tabla 18: *Prueba de escritorio.*

	a	**b**	**c**
Línea 1	2		
Línea 2		5	
Línea 3			7
Línea 4			4
Línea 5		5	
Línea 6	8		
Línea 7			0
Línea 8	**8**	**5**	**0**

Rangos

Cuando se realizan validaciones para un valor o se utilizan límites debemos considerar el uso de operadores de relación. La recta numérica es útil recordar cuando se debe establecer valores límites que se incluyen en un rango determinado. En la figura 11, se considera un rango aceptable para las horas del día. Se considera de 00h00 a 24h00 (distinguiendo 00h00 y 24h00 separadamente). Para

valores menores a cero o mayores a 24 se considera un mal ingreso. Para valores mayores e iguales a cero y menores e iguales a 24 se considera un buen ingreso.
Acentuamos el uso de los operadores lógicos cuando se unen dos expresiones.

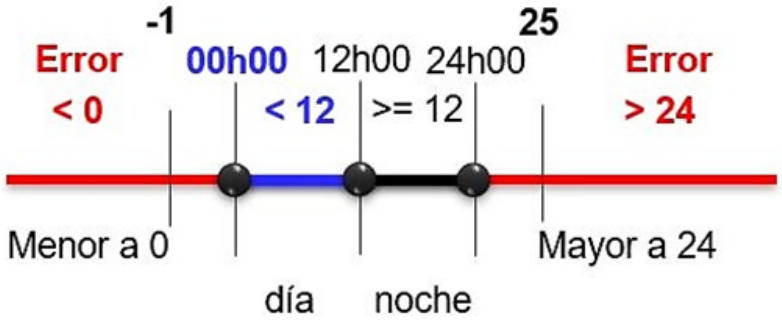


Figura 71: *Uso de la recta numérica para determinar operadores relacionales.*

Aquí debemos recordar la teoría de proposiciones que representan las dos expresiones a validar. Recordamos el álgebra de Boole para evaluar las operaciones con los operadores lógicos. Recordamos también las tablas de verdad para validar el resultado lógico final de la expresión en la condición. Y finalmente recordamos el teorema de Morgan para demostrar que la negación de una expresión puede transformarse a una expresión similar siguiendo las reglas del teorema, la negación permite cambiar el operador lógico y los operadores relacionales internos. A continuación, en la tabla 19, se muestra la capacidad de representar un mismo código de formas distintas para un mismo problema. Los diez bloques realizan lo mismo.

Tabla 19: *Bloque de estructuras similares en su ejecución.*

Si (hora >= 0) y (hora <=24) Escribir (“ Buen ingreso ”) Sino Escribir (“ Mal ingreso ”)	Si (hora <0) ó (hora >24) Escribir (“ Mal ingreso ”) Sino Escribir (“ Buen ingreso ”)
Si (hora >-1) y (hora <25) Escribir (“ Buen ingreso ”) Sino Escribir (“ Mal ingreso ”)	Si (hora <= -1) ó (hora >= 25) Escribir (“ Mal ingreso ”) Sino Escribir (“ Buen ingreso ”)
Si negado((hora >-1) y (hora <25)) Escribir (“ Mal ingreso ”) Sino Escribir (“ Buen ingreso ”)	Si negado((hora <0) ó (hora >24)) Escribir (“ Buen ingreso ”) Sino Escribir (“ Mal ingreso ”)
Si (negado(hora >-1) ó negado(hora <25)) Escribir (“ Mal ingreso ”) Sino Escribir (“ Buen ingreso ”)	Si (negado(hora <0) y negado(hora >24)) Escribir (“ Buen ingreso ”) Sino Escribir (“ Mal ingreso ”)
Si ((hora <= -1) ó (hora >=25)) Escribir (“ Mal ingreso ”) Sino Escribir (“ Buen ingreso ”)	Si ((hora >= 0) y (hora <= 24)) Escribir (“ Buen ingreso ”) Sino Escribir (“ Mal ingreso ”)

El uso de los operadores lógicos permite unir dos o más expresiones.
El operador lógico *Or* permite unir dos rangos diferentes. El operador *And* permite determinar el cumplimiento de dos condiciones que se interceptan (Ver Fig. 72).

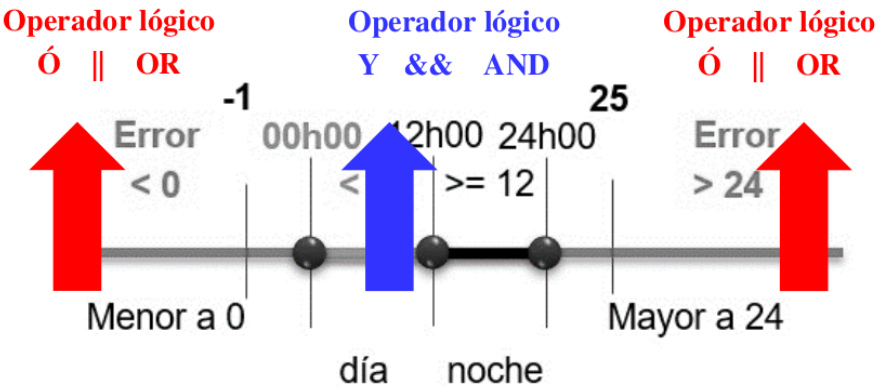


Figura 72: *Uso de la recta numérica para determinar operadores lógicos.*

Codifique un programa que permita el ingreso de un valor entero que represente la hora del día (considere el cero y el 24). Definiendo un mensaje de salida si el valor es mayor e igual a cero y menor e igual a 24, mostrando "Buen ingreso", caso contrario "Mal ingreso".

Código 9: rangos1.py

```
# Comentario: rangos1.py es el nombre del archivo
# Comentario Rango versión 1
hora = int(input("Ingrese un valor: "))
if (hora >= 0 and hora <= 24):
    print("Buen ingreso")
else:
print("Mal ingreso")
```

El resultado en pantalla sería:

```
Ingrese un valor: -1
Mal ingreso
```

Para este ejercicio, tenemos todas las versiones que nos presenta la tabla 19, el desafío es probar con todas. Si observamos la primera línea del código 9, encuentras el símbolo del numeral "#" que permite colocar comentarios, texto de referencia para el programador, no interfiere con el código escrito. Si lo omites (puedes borrarlo) ocasiona un problema de sintaxis. La segunda línea es similar a lo que se ha aprendido, ingresar un valor de entrada. La tercera y última línea es una estructura selectiva llamada condicional sí doble, debido a que luego de que valida la condición, si ésta es verdadera (if (hora $>= 0$ and hora $<= 24$):), se muestra el mensaje por verdadero, caso contrario (else:) se muestra el mensaje por falso. Se prueban dos bloques de códigos más de la tabla 19. Las estructuras condicionales se verán más adelante.

Código 10: rangos2.py

```
# Comentario: rangos2.py es el nombre del archivo
# Comentario Rango versión 2
hora = int(input("Ingrese un valor: "))
if (hora <0 or hora >24):
    print("Mal ingreso")
else:
print("Buen ingreso")
```

El resultado en pantalla sería:

```
Ingrese un valor: 0
Buen ingreso
```

Si observamos la tercera línea del código 10, se nota que se ha utilizado otro bloque de código de la tabla 19 y nuevamente funciona bien, para el código 9 se usó el operador and, para el código 10 se usa el operador booleano *or*. Observe detenidamente la ubicación de los mensajes.

Código 11: rangos3.py

```
# Comentario: rangos3.py es el nombre del archivo
# Comentario Rango versión 3
hora = int(input("Ingrese un valor: "))
if (not (hora <0) and not (hora >24)):
    print("Buen ingreso")
else:
print("Mal ingreso")
```

El resultado en pantalla sería:

```
Ingrese un valor: 24
Buen ingreso
```

Finalmente, se ha utilizado un tercer bloque de las propuestas presentadas en la tabla 19, nuevamente funciona, lo que debe observar son los mensajes de salida para cuando la condición se cumple por verdadero y cuando ésta se cumple por falso.

3.2.3 ESTRUCTURAS ALGORÍTMICAS

Estructuras selectivas

Las estructuras selectivas permiten escoger una opción (camino o ruta) luego de la validación de una condición, se las conoce como bifurcaciones. Se las utiliza en los programas cuando se desea validar un valor y realizar un conjunto de acciones para dicho valor, o lo contrario, no realizar acción alguna o ejecutar otro conjunto de instrucciones. Hay muchos ejemplos que en la realidad se pueden asemejar a problemas cotidianos.

Codifique un programa que muestre un mensaje cuando un usuario ingresa su edad. Si el usuario es mayor e igual a 18, se muestra un mensaje de "Es mayor de edad". Caso contrario no muestra mensaje alguno.

Código 12: si_simple.py

```
# Comentario: si_simple.py es el nombre del archivo
# Comentario Estructura sí simple
edad = int(input("Ingrese su edad: "))
if (edad >= 18):
    print("Es mayor de edad")
```

El resultado en pantalla sería:

```
Ingrese su edad: 56
Es mayor de edad
```

Si se observa el código en la tercera línea se encuentra la estructura selectiva llamada *sí simple*, se usa la palabra reservada *if*, luego la condición donde se observa los operadores de relación "*mayor e igual*", si recordamos la teoría de proposiciones, ésta es una y se puede unir otras expresiones utilizando operadores lógicos (booleanos). Se observa además el uso de los dos puntos al final de la condición y lo más característico de Python, el uso de la tabulación, que agrupa una o más líneas que significaría que se ejecutan por verdadero.

Codifique un programa que muestre un mensaje cuando un usuario ingresa su edad. Si el usuario es mayor e igual a 18, se muestra un mensaje de "Es mayor de edad", caso contrario se muestra un mensaje "Es menor de edad".

Código 13: si_doble1.py

```
# Comentario: si_doble1.py es el nombre del archivo
# Comentario Estructura sí doble
edad = int(input("Ingrese su edad: "))
if (edad >= 18):
    print("Es mayor de edad")
else:
    print("Es menor de edad")
```

El resultado en pantalla sería:

```
Ingrese su edad: 12
Es menor de edad
```

Si se observa la tercera línea se utiliza el mismo código del ejercicio anterior, se ha agregado un mensaje cuando la condición no es verdadera.

Codifique un programa que muestre si el número 7 es primo o no.

Código 14: si_simples.py

```
# Comentario: si_simples.py es el nombre del archivo
# Comentario Estructura varios sí simple
num = int(input("Ingrese el número 7: "))
Contador = 0
if (num% 1 == 0):
    Contador = Contador + 1
if (num% 2 == 0):
    Contador = Contador + 1
if (num% 3 == 0):
    Contador = Contador + 1
if (num% 4 == 0):
    Contador = Contador + 1
if (num% 5 == 0):
    Contador = Contador + 1
if (num% 6 == 0):
    Contador = Contador + 1
if (num% 7 == 0):
    Contador = Contador + 1
if (Contador <= 2):
    print("El número ", num, "es Primo")
else:
```

```
    print("El número ", num, "es No Primo")
print("Contador contiene ", Contador)
```

El resultado en pantalla sería:

```
Ingrese el número 7: 7
El número 7 es Primo
Contador contiene 2
```

Se observa en este código, que en la tercera línea se tiene el uso de una variable llamada contador (el nombre usado «que puede ser cualquiera» en el programa usa la primera letra con mayúscula) y su utilidad es de aumentar en 1 si se cumple la condición.

Si sigue la secuencia del programa el número 7 cumplirá en dos condiciones y por verdadero el contador llamado *Contador* sumará un uno al valor que guarda, la primera vez será 0 + 1, y la segunda vez será 1 + 1.

Y tiene coherencia ya que el número 7 es primo por naturaleza y todo número primo lo es cuando es divisible para sí mismo y para la unidad. El operador % representa el módulo o resto de una operación.

Este programa puede ser mejorado en la sección de estructuras repetitivas, por ahora sólo funciona con el número 7 y, menores a 7 y mayor a cero.

Codifique un programa que permita el ingreso de dos números no iguales e identifique el mayor de ellos.

Código 15: si_doble2.py

```
# Comentario: si_doble2.py es el nombre del archivo
# Comentario Estructura sí doble
a = int(input("Ingrese el número a: "))
b = int(input("Ingrese el número b: "))
if ( a >b ):
    print(a, "es mayor a ", b)
else:
    print(b, "es mayor a ", a)
```

El resultado en pantalla sería:

```
Ingrese el número a: 34
Ingrese el número b: 21
34 es mayor a 21
```

Para este programa el usuario debe ingresar dos números no iguales, y funciona. Pero ¿qué ocurre si se ingresan dos números iguales?

Codifique un programa que permita el ingreso de dos números (incluso iguales) e identifique el menor de ellos.

Código 16: si_doble3.py

```
# Comentario: si_doble3.py es el nombre del archivo
# Comentario Estructura sí doble
a = int(input("Ingrese el número a: "))
b = int(input("Ingrese el número b: "))
if ( a == b ):
    print(a, "es igual a ", b)
else:
    if ( a >b ):
      print(a, "es mayor a ", b)
    else:
      print(b, "es mayor a ", a)
```

El resultado en pantalla sería:

```
Ingrese el número a: 22
Ingrese el número b: 22
22 es igual a 22
```

Si observamos la diferencia entre el código 15 y el 16, se nota que se pregunta (o valida) por si los dos números ingresados son iguales, y luego por falso se procede a colocar otra estructura sí doble para determinar si es mayor *a* o *b*. Debe observar que se van anidando los *if* cada vez que se supera una validación o una condición. Va tomando la forma de una escalera si usamos una referencia visual con el uso de las tabulaciones o identaciones. Otra observación es el uso de la coma y el más dentro del *print()*, si usted usa la coma, en la siguiente cadena de caracteres no es necesario utilizar espacios para separar variables siguientes, ejemplo código 16, en el código 17, se usa la función *str()*, que transforma el valor entero de a en una cadena de caracteres o simplemente texto, pero para que pueda ser visible junto a la cadena, se debe transformarlo usando *str()*, y debe utilizarse espacios dentro de la siguiente cadena para que no se unan (junten), el símbolo + entonces significa concatenar una o más cadenas de caracteres.

Código 17: si_doble4.py

```
# Comentario: si_doble4.py es el nombre del archivo
# Comentario Estructura sí doble
a = int(input("Ingrese el número a: "))
b = int(input("Ingrese el número b: "))
if ( a == b ):
    print(str(a) + "es igual a " + str(b))  note este cambio
    print(a, "es igual a ", b)  quite el primer numeral
else:
    if ( a >b ):
      print(a, "es mayor a ", b)
    else:
      print(b, "es mayor a ", a)
```

El resultado en pantalla sería:

```
Ingrese el número a: 22
Ingrese el número b: 22
22es igual a22
```

Observe que al usar *str()*, se transforma el valor numérico a un texto, y requerirá el uso de espacios vacíos antes y después de la cadena, note el resultado en pantalla que está unido *22es* y *a22*, por la falta de espacios dentro de la cadena. Una manera de colocar las estructuras selectivas de forma consecutiva una validación tras otra, según se lo requiera o deba de cumplirse condiciones, es utilizando los sí anidados, en Python, el código 17 se mejora utilizando la palabra reservada *elif*.

Codifique un programa que permita el ingreso de tres números (incluso iguales) e identifique el mayor de ellos.

Código 18: si_anidado1.py

```
# Comentario: si_anidado1.py es el nombre del archivo
# Comentario Estructura sí anidado
a = int(input("Ingrese el número a: "))
b = int(input("Ingrese el número b: "))
c = int(input("Ingrese el número c: "))
if ( (a == b) and (a == c) ):
    print(str(a) + .es igual a` str(b) + .eigual a` str(c))
elif ( a >b ):
```

```
    if ( a >c):
      print(a, "es mayor a ", c)
    else:
      print(c, "es mayor a ", a)
elif ( b >c ):
    print(b, "es mayor a ", c)
else:
    print(c, "es mayor a ", b)
```

El resultado en pantalla sería:

```
Ingrese el número a: 2
Ingrese el número b: 3
Ingrese el número c: 1
3 es mayor a 1
```

Para este ejercicio, retomamos la prueba de escritorio y los resultados posibles al ingresar los valores de acuerdo con un orden establecido.

Tabla 20: *Posibles resultados al evaluar el programa con los valores ingresados.*

a	b	c	Salida
1	2	3	c con 3
1	3	2	b con 3
3	1	2	a con 3
3	2	1	a con 3
2	1	3	c con 3
2	3	1	b con 3

Las estructuras selectivas permiten la bifurcación del flujo de datos o recorrido en dos posibles respuestas (o más) que se validan por una condición, una respuesta por verdadero y una respuesta por falso.

DESAFÍO PARA LAS ESTRUCTURAS SELECTIVAS

Codifique un programa que permita el ingreso de un valor entero y que represente la edad de una persona. Si se ingresa un valor menor a cero y mayor a 100 se muestra un mensaje de “Incorrecto”. Si se ingresa un valor igual a cero se muestra un mensaje de “Bebé”. Si se ingresa un valor mayor a cero y menor a doce se muestra un mensaje de “Es niño/a”. Si se ingresa un valor mayor igual a doce y menor a dieciocho se muestra un mensaje de “Es joven”. Si se ingresa un valor mayor igual a dieciocho y menor a sesenta se muestra un mensaje de “Es adulto”. Si se ingresa un valor mayor igual a sesenta y menor igual a cien se muestra un mensaje de “Es adulto mayor”.

Intente utilizar la estructura con elif, recuerde que hay muchas formas de representar el problema mediante un algoritmo y también en codificación.

Código 19: si_anidado2.py

```
# Comentario: si_anidado2.py es el nombre del archivo
# Comentario Estructura condicional
edad = int(input("Ingrese su edad: "))
if ( (edad <0) and (edad >100) ):
    print("Incorrecto")
elif ( edad == 0 ):
    print("Bebé")
elif ( (edad >0) and ( edad <12)):
    print("Es niño/a")
elif ( (edad >= 12) and ( edad <18)):
    print("Es joven")
elif ( (edad >= 18) and ( edad <70)):
    print("Es adulto")
elif ( (edad >= 70) and ( edad <= 100)):
    print("Es adulto mayor")
```

El resultado en pantalla sería:

```
Ingrese su edad: 13
Es joven
```

Codifique un programa que identifique si un número es Armstrong o no de tres cifras. Un número de *n* dígitos se llama Armstrong si la suma de sus dígitos elevados a la *n* es igual al número mismo. Por ejemplo, un número de 3 cifras *abc* es Armstrong si es igual a $a^3 + b^3 + c^3$. Verifique que 371 es un número de Armstrong y encuentre otro de 3 cifras.

Primero se debe entender el problema, un valor de tres cifras significa un número entre 100 a 999. Luego se debe separar sus cifras en unidad, decena y centena. Finalmente elevar al cubo cada una de las cifras y sumarlas. Si al sumar volvemos a obtener el número inicial es Armstrong. De acuerdo con el orden de la jerarquía de operaciones aritméticas se puede obtener varias versiones del programa. Presentamos un primer código:

Código 20: armstrong1.py

```
# Comentario: armstrong1.py es el nombre del archivo
# Comentario versión 1 de Armstrong de tres cifras
n = int(input("Ingrese un número de tres cifras: "))
```

```
u = int(n% 10)
d = int((n // 10)% 10)
c = (n // 100)
m = u*u*u + d*d*d + c*c*c
if (m == n):
    print("Es Armstrong")
else:
    print("No es Armstrong")
```

El resultado en pantalla sería:

```
Ingrese un número de tres cifras: 407
Es Armstrong
```

Observe el uso de las dos // barras de divisiones, representan la división entera, es decir, se obtiene el valor entero de la división sin decimales. Otra forma es invocar al método o función *int()*. Un segundo código se presenta:

Código 21: armstrong2.py

```
# Comentario: armstrong2.py es el nombre del archivo
# Comentario versión 2 de Armstrong de tres cifras
n = int(input("Ingrese un número de tres cifras: "))
u = (n% 100)% 10
d = int((n% 100) // 10)
c = (n // 100)
m = u*u*u + d*d*d + c*c*c
if (m == n):    print("Es Armstrong")
else:
    print("No es Armstrong")
```

El resultado en pantalla sería:

```
Ingrese un número de tres cifras: 307
Es Armstrong
```

Pruebe el código para los números 371 y 153. Si puedes obtener otra forma de presentar la respuesta utilizando otro código vas por buen camino.

3.2.4 ESTRUCTURAS REPETITIVAS

En el lenguaje de Programación de Python, solo existen dos estructuras repetitivas, estas son, la estructura repetitiva *while* y la estructura repetitiva *for*. Se les conoce como bucles o lazos. La estructura *while* repite el bloque de sentencias mientras sea verdadera la condición. La estructura *for* es útil cuando conocemos la cantidad inicial y final de repeticiones que debemos realizar.

Estructuras selectivas

Las estructuras selectivas permiten escoger una opción (camino o ruta) luego de la validación de una condición, se las conoce como bifurcaciones. Se las utiliza en los programas cuando se desea validar un valor y realizar un conjunto de acciones para dicho valor, o lo contrario, no realizar acción alguna o ejecutar otro conjunto de instrucciones. Hay muchos ejemplos que en la realidad se pueden asemejar a problemas cotidianos.

Codifique un programa que permita presentar en pantalla los números del 1 al 15.

Código 22: while1.py

```
# Comentario: while1.py es el nombre del archivo
# Programa que presenta los números del 1 al 15
x=1
while x<=15:
    print(x, end=‘ ’)
    x=x+1
```

El resultado en pantalla sería:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Veamos otro ejemplo con la estructura *while*.

Codifique un programa que permita presentar en pantalla los números del 1 al 15 de forma descendente.

Código 23: while2.py

```
# Comentario: while2.py es el nombre del archivo
# Programa que presenta los números del 1 al 15 de forma descendente
x=15
```

```
while x>=1:
    print(x, end=‘ ’)
    x=x-1
```

El resultado en pantalla sería:

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Te desafiamos a que quites end=‘ ’ de la estructura *while*. Debes observar que los números se desplazan hacia abajo.

Codifique un programa que permita presentar en pantalla los números del 1 al 15, utilizando la estructura *for*.

Código 24: for1.py

```
# Comentario: for1.py es el nombre del archivo
# Programa que presenta los números del 1 al 15
for x in range(15):
    print(x, end= ’ ’)  primera salida
print()
for x in range(1,15,1):
    print(x, end= ’ ’)  segunda salida
for x in range(1,16,1):
    print(x, end= ’ ’)  tercera salida
```

El resultado en pantalla de las tres salidas sería:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Puedes notar que la primera salida del lazo *for* el listado de números va del cero al catorce, entendiendo que la variable x inicia siempre en cero y que esta termina un valor anterior al definido en *range*. En cambio, en la segunda salida, puedes observar que *range* tiene otra forma de escribirse, ahora con tres parámetros, el primero, el 1, es el inicio definido, el segundo, el 15, es el valor final sin incluir, y el tercero, el 1, es el incremento o el paso de iteración a iteración. Para los casos anteriores, ninguno es lo que se desea, mostrar los números del 1 al 15.

Entonces, tratemos de entender esta estructura repetitiva, los valores en el range, deben ser 1, 16, 1, entendiendo que el valor de 16 significa que no llega a dicho número sino llega a uno antes.

Codifique un programa que permita presentar en pantalla los números del 1 al 15 de forma descendente.

Código 25: for2.py

```
# Comentario: for2.py es el nombre del archivo
# Programa que presenta los números del 1 al 15 de forma descendente
for x in range(15,0,-1):
    print(x, end= ' ')
```

El resultado en pantalla sería:

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

DESAFÍO PARA LAS ESTRUCTURAS REPETITIVAS

Codifique un programa que permita el ingreso de 4 notas de estudiantes sobre 20 puntos, indicar cuántos aprobaron si para aprobar la nota debe ser mayor a 14.

Este programa requerirá del uso de la estructura repetitiva *while* y de una estructura selectiva *if*. Usa una variable *nota*, una variable *x* que hace de contador y una variable *total*.

Código 26: while3.py

```
# Comentario: while3.py es el nombre del archivo
# Programa que presenta cuántos estudiantes aprobaron
x=0
total=0
while x<4:
    nota=int(input("Ingrese la nota: "))
    if nota>14:
       total=total+1
    x=x+1
print("Han aprobado",total)
```

El resultado en pantalla sería:

```
Ingrese la nota: 19
Ingrese la nota: 12
Ingrese la nota: 20
Ingrese la nota: 19
Han aprobado 3
```

Si observas las dos primeras variables, tanto *x* como *total* se inicializan con cero, es decir la variable *x* se la usa como un contador, y la variable *total* almacenará la cantidad de aquellas notas que son mayores a 14, como indica el *if*.

Codifique un programa que permita el ingreso de 4 números cualquiera y muestre en pantalla la suma de los últimos dos.

Este programa requerirá del uso de la estructura repetitiva *for* y de una estructura selectiva *if*. Usa una variable suma que será usada como acumulador y la variable *f* de iteración.

Código 27: for3.py

```
# Comentario: for3.py es el nombre del archivo
# Programa que presenta la suma de los dos últimos números
suma=0
for f in range(4):
    valor=int(input("Ingrese un valor: "))
    if f>1:
       suma=suma+valor
print("La suma de los últimos 2 valores es: ")
print(suma)
```

El resultado en pantalla sería:

```
Ingrese un valor: 12
Ingrese un valor: 2
Ingrese un valor: 5
Ingrese un valor: 4
La suma de los últimos 2 valores es:
9
```

Si observas la estructura repetitiva *for* dentro de ella, está el ingreso del valor (dado por el usuario) y la condición del *if* indica mayor a 1, eso quiere decir que no cuenta la posición 0 ni 1, sino que a partir del mayor a 1, es decir 2 y 3 (porque son 4) se sumarán en la variable *suma*. Recuerda del 0 al 3 hay cuatro ingresos. Considerar al cero implica no llegar al valor final indicado, 4. En este lenguaje como en otros la posición cero es tomada en cuenta.

3.2.5 ARREGLOS

En el lenguaje de Programación de Python, existen las estructuras que conforman un grupo de elementos que pueden tener un mismo nombre e invocados por valor llamado índice, a estos elementos se los conoce como arreglos (vectores o arrays). Pero ahora pasan a tener un nombre más genérico conocido como listas. Más adelante encontrarás también otras estructuras que por su orden o ubicación toman nombres como, diccionarios (que son listas asociadas), tuplas y conjuntos de datos.

Codifique un programa que permita presentar en pantalla los números ingresados por teclado del 1 al 4.

Código 28: arreglos1.py

```
# Comentario: arreglos1.py es el nombre del archivo
# Programa que presenta los números ingresados del 1 al 4 usando arreglos
arreglo1 = []
print(arreglo1)
index = 4
for i in range(index):
    arreglo1.append(i)
    arreglo1[i] = input()
print(arreglo1)
print()
for i in range(len(arreglo1)):
    print(arreglo1[i], end=' ')
```

El resultado en pantalla sería:

```
[ ]
1
2
3
4
['1', '2', '3', '4']

1 2 3 4
```

A simple vista la declaración de un arreglo es sencilla. Debes recordar que en la declaración podemos indicar la cantidad de elementos (espacios de memoria) a disponer en el problema (programa) pero en muchos casos es probable que no sabemos cuántos espacios requerirás. Para eso debemos comprender que, a estas estructuras, el lenguaje de programación Python tiene asignado funciones

propias por su naturaleza. Un ejemplo es *append()* que observas en la primera línea dentro del lazo *for*.

En este ejemplo, la variable que va dentro del *range* llamada *index* se le asigna el valor de 4, esta práctica de usar variables en vez de colocar el número directamente en las estructuras debes recordar, ya que así cuando se requiera que el programa sea de almacenamiento de más valores, sólo cambiarás el valor de *index* en la tercera línea.

Si observas `arreglo1[i] = input()`, va a permitir el ingreso de los números por el usuario y luego se los presentará a todos de golpe. Por eso después del valor de ingreso, 4, se ven con los corchetes y comillas simples cada elemento.

El segundo lazo *for* es interesante su análisis, dentro del *range*, encuentras una función llamada *len()* que determina la longitud del arreglo1, y devuelve el valor de la cantidad de elementos que contiene esa estructura (arreglo1), es decir el valor de 4. Ese valor de cuatro quiere indicarnos que el arreglo1 va de la posición 0 a la posición 3, no llega al 4. Finalmente, la salida se presenta. 1 2 3 4.

DESAFÍO PARA ARREGLOS

Codifique un programa que permita presentar una palabra correspondiente a un color del idioma español al inglés.

Por ejemplo, para este programa usaremos dos arreglos almacenando palabras en español y en inglés y en un orden correspondiente.

Código 29: arreglos2.py

```
# Comentario: arreglos2.py es el nombre del archivo
# Programa que presenta un traductor de colores de español a inglés
espaniol = ["verde", "negro", "amarillo", "rojo", "azul"]
ingles = ["green", "black", "yellow", "red", "blue"]

colore = input("Ingrese un nombre de color en español: ")
encuentra = colore in espaniol
if encuentra == True:
    ubica = espaniol.index(colore)
    respuesta = ingles[ubica]
else:
    respuesta = "No está ese color traducido"
print("Tu resultado es:",encuentra)
print("La traducción sería:",respuesta)
```

El resultado en pantalla sería:

```
Ingrese un nombre de color en español: limón
Tu resultado es: False
La traducción sería: No está ese color traducido
```

Si realizamos un segundo ingreso, este sería:

```
Ingrese un nombre de color en español: azul
Tu resultado es: True
La traducción sería: blue
```

Analicemos este código. Puedes notar que las dos primeras líneas se almacenan los arreglos de palabras en español y en inglés, estas deben estar ubicadas en el mismo orden, las palabras originales y sus traducciones. En la variable llamada *colore*, se establece el ingrese de valores por el usuario. La variable *encuentra*, detecta si *colore* se encuentra en el arreglo *espaniol*.

En la estructura *if*, se valida si la variable *encuentra* es verdadera (True), si esta lo es, se ubica la posición del color en el arreglo *espaniol*, y se detecta la ubicación en el segundo arreglo mediante el índice *ubica* en el arreglo ingles asignado a *respuesta*.

En caso de ser falso (False), la variable *respuesta*, se le asigna un mensaje de “No está ese color traducido”.

3.2.6 MATRICES

Trabajar con una gran cantidad de datos nos permite hacer tratamientos complejos y obtener mejores resultados. El uso de matrices en Python guarda la misma similitud en correspondencia con las matrices que se estudian en Álgebra Lineal. Para ello el uso de filas (renglones) o columnas permitirán establecer una lógica de almacenamiento y posición dentro de la matriz. Finalmente, recuerda que al ser un tipo de estructura compleja (que proviene de datos simples) tendrá características y funciones que permiten usar el lenguaje.

DESAFÍO PARA MATRICES

Codifique un programa que permita presentar en pantalla una matriz de 4x4.

Código 30: matriz1.py

```
# Comentario: matriz1.py es el nombre del archivo
# Programa que presenta una matriz de 4x4
matriz1 = [[],[],[],[]]
print(matriz1)
index = 4
for i in range(index):
    matriz1.append(i)
    for j in range(index):
       matriz1[i].append(j)
       matriz1[i][j] = input()
       print("[",i,"][",j,"]:",matriz1[i][j])
print()
for i in range(index):
    for j in range(index):
       print("\t" +matriz1[i][j], end=' ')
    print()
```

El resultado en pantalla sería:

```
[[], [], [], []]
1
[ 0 ][ 0 ]: 1
2
[ 0 ][ 1 ]: 2
3
[ 0 ][ 2 ]: 3
```

```
4
[ 0 ][ 3 ]: 4
5
[ 1 ][ 0 ]: 5
6
[ 1 ][ 1 ]: 6
7
[ 1 ][ 2 ]: 7
8
[ 1 ][ 3 ]: 8
9
[ 2 ][ 0 ]: 9
10
[ 2 ][ 1 ]: 10
11
[ 2 ][ 2 ]: 11
12
[ 2 ][ 3 ]: 12
13
[ 3 ][ 0 ]: 13
14
[ 3 ][ 1 ]: 14
15
[ 3 ][ 2 ]: 15
16
[ 3 ][ 3 ]: 16

  1   2   3   4
  5   6   7   8
  9   10  11  12
  13  14  15  16
```

De forma similar con el uso de arreglos, tenemos en la primera línea la estructura de la matriz, los primeros corchetes representan a la matriz, y los cuatro corchetes interiores representan a las cuatro filas. La forma de incrementar elementos será mediante *append()*, aquí podemos decirte que existen paquetes de funciones creadas por la comunidad de Python para optimizar códigos y resolver estos primeros problemas con el uso de dichas funciones, entre esos paquetes tenemos a Numpy 7[7] o Pandas 8[8] que con las definiciones de sus códigos de funciones o sintaxis podremos superar estas declaraciones. Por ahora *append()* será de mucha ayuda. La segunda línea presenta la matriz vacía. A continuación, la variable *index* con el valor de 4 nos va a permitir controlar los

[7]Disponible en https://numpy.org/

[8]Disponible en https://pandas.pydata.org/

dos lazos *for* para la variable *i* y la variable *j*.

Recuerda que el proceso con *append()* es "crear el espacio siguiente y asignar", por eso puede ver dos veces a *append()*, dentro del lazo *for* para *j* es debido a que está creando una columna y luego se usa, y la primera línea dentro del lazo *for* para *i*, *append()* crea una fila y le da paso al lazo *for* para *j*, para que la use. Finalmente cuando ya está creado el espacio de la fila y de la columna, se procede a asignarlo en `matriz1[i][j] = input()`.

El estilo de presentación de cada ingreso lo realiza el *print()*, puedes hacer uso de la coma entre los textos en comillas y las variables, también puedes usar el operador de concatenación el +, y caracteres especiales como el tabulador o espaciado (llamado sangría), entre comillas, "\t".

3.2.7 FUNCIONES

En el lenguaje de Python, las funciones permiten agrupar sentencias que representan aquellos procedimientos que el programa deba realizar. Así como existen funciones en Excel, o funciones trigonométricas (Coseno, Tangente, etc.), las funciones permiten realizar una tarea independiente del programa y que ayuda al programa a extraerla del código principal. Una función puede ser conocida como un método de hacer algo, o puedes verla como un programa que anteriormente ha funcionado y ahora lo reduces a un bloque, con un nombre definido, sus parámetros (si son o no necesarios), sentencias que involucran procesos y un valor de retorno si se desea. Finalmente, una función permite dividir el problema y pode ser usada en otros programas.

DESAFÍO PARA FUNCIONES QUE NO RETORNAN VALORES

Codifique un programa que permita presentar en pantalla una matriz de 4x4.

Código 31: function1.py

```
# Comentario: function1.py es el nombre del archivo
# Programa que presenta funciones que no retornan valores
def funcion1a():
    print("Hola")

def dibujarcuadrado():
    print("*********")
    print("*       *")
    print("*       *")
    print("*********")

def funcion1b():
```

```
    print("Adiós")

funcion1a()
dibujarcuadrado()
funcion1b()
```

El resultado en pantalla sería:

```
Hola
*********
*       *
*       *
*********

Adiós
```

Puedes observar que en este código tenemos tres funciones, cada una de ellas se las conoce como funciones que no retornan valor no requieren de parámetros. Es decir "hacen algo" pero no devuelven valor alguno a otra variable, que no es lo mismo que presentar algo, ya que la salida se presenta, un "hola", un cuadrado de asteriscos, y un "adiós".

Por eso debes comprender, devolver algo es como tener $y = tan(x)$ donde la función tangente devuelve un valor a y. Asimismo la llamada o invocación de la función es colocar su nombre seguido de los paréntesis.

DESAFÍO PARA FUNCIONES QUE RETORNAN VALORES

Código 32: function2.py

```
# Comentario: function2.py es el nombre del archivo
# Programa que presenta una función que retorna un valor
def main():
    base = int(input("Ingrese la base: "))
    expo = int(input("Ingrese el exponente: "))
    potencia = poten(base, expo)
    print("La potencia de",base,"a la",expo,"es",potencia)
    fin()

def poten(ba, ex):
    acum = 1
    for i in range(ex):
       acum = acum * ba
```

```
    return acum

def fin():
    print("Finalizó el programa")

main()
```

El resultado en pantalla sería:

```
Ingrese la base: 2
Ingrese el exponente: 5
La potencia de 2 a la 5 es 32
Finalizó el programa
```

Puedes notar que hay tres funciones, y una llamada desde la función *main()*, que sería realmente el programa que como has visto la habilidad de colocar todo en funciones, este programa se lo ha encapsulado en una sola línea. De las tres funciones que observas (que primero se escriben y luego sus invocaciones), dos de ellas no retornan nada sino que hacen algo *(main()* y *fin())* y algo muy interesante es que *main()* llamada a las otras dos.

La función`poten(ba, ex)` es la que hace el proceso de obtener la potencia de un número entregándole a ella un valor de base y un valor de exponente con ello, acumula *expo* veces la multiplicación de la *base*. Como es el proceso de obtener la potencia de un número. La ayuda del lazo *for*, permite la acumulación. El acumulador se le asigna 1, debido a que las acumulaciones de productos o cocientes el valor de 1 es requerido para que no afecten a las otras repeticiones (multiplicaciones o divisiones sucesivas). Recuerda que si en cambio, esto fuera la acumulación de valores por medio de sumas o restas, el valor de la variable debería ser cero para que al sumar o restar el cero no afecte a la operación aritmética.

Finalmente, la variable *potencia* es la encargada de recibir lo que retorna la función `poten(base, expo)`, comprende esa línea con la última línea de la función *poten()*, en ella se muestra la sentencia *return*, que significa que retorna el contenido de *acum*, a quién le esté invocando desde otro lugar del programa, o dentro de ella misma (por ejemplo recursividad). Ánimo, la construcción de funciones realmente es sencillo cuando ya elabores programas y como estos funcionan, los podrás usar en otros problemas guardándolos como paquetes de funciones.

3.2.8 LENGUAJE MATLAB

MATLAB es el entorno software más sencillo y productivo para ingenieros y científicos, puedes conseguirlo desde el siguiente enlace[9]. Puedes hacer un curso gratuito en MathWorks en este enlace `https://la.mathworks.com/learn/tutorials/matlab-onramp.html`.

MATLAB distingue minúsculas y mayúsculas. Esto quiere decir que a y A representan variables diferentes. Introducción de matrices. Los elementos de un renglón de una matriz se separan por espacios y/o comas, y con ello se producen las columnas, al colocar ";" podemos distinguir un vector (fila) de las demás.

Por ejemplo, la siguiente línea (una matriz): $A = [1\ 2\ 3\ ;4\ 5\ 6\ ;7\ 8\ 9]$ produce la siguiente información en pantalla:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

De igual manera si escribimos de esta forma (dando saltos de línea):

$A = [1\ 2\ 3\ ;$

$4\ 5\ 6\ ;$

$7\ 8\ 9\ ]$

Se produce la misma respuesta, es decir, si escribes de forma horizontal o vertical con saltos de línea luego del punto y coma, se produce el mismo resultado.

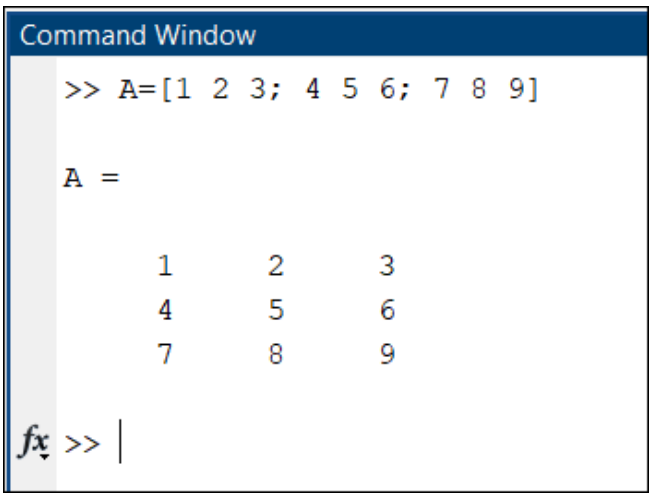


Figura 73: *Salida de una matriz 3×3 en Matlab.*

[9]Disponible en `https://la.mathworks.com/products/matlab.html`

Ahora si generamos una matriz B como esto: $B = [3;\ 6;\ 1]$ se produce la siguiente salida:

$$B = \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix}$$

Entonces podemos darnos cuenta de que, el punto y coma representa el final de una línea o lo que podemos llamar una *fila* en una matriz.

```
>> B = [3; 6; 1]

B =

     3
     6
     1

fx >>
```

Figura 74: *Salida de una matriz 3×1 en Matlab.*

Por ejemplo, si deseamos utilizar una notación para formar las submatrices y las matrices aumentadas, podemos seguir los siguientes ejemplos.

```
>> A=[1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> f=A(2,3)

f =

     6

fx >>
```

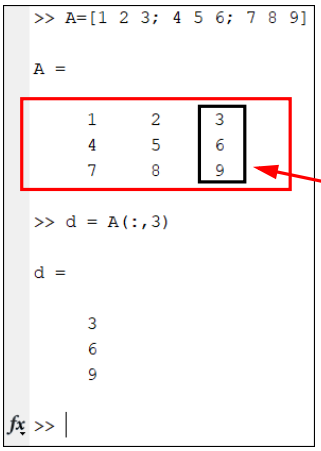
Si digitamos: $f = A(2,3)$

f es el elemento en el segundo renglón, tercera columna de A.

```
>> A=[1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> d=A(3,:)

d =

     7     8     9

fx >>
```

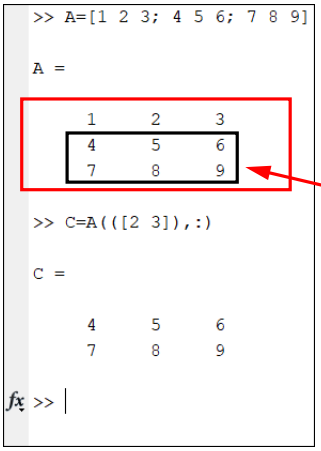
Si digitamos: $d = A(3,:)$

d es el tercer renglón de A.



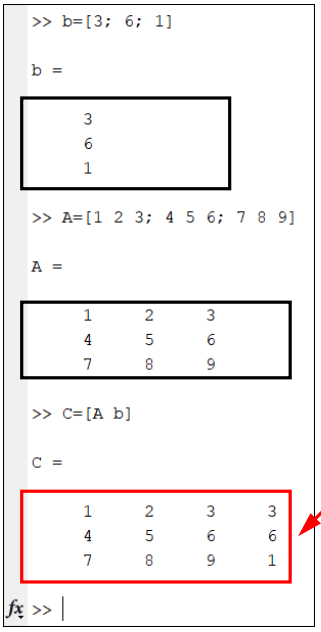
Si digitamos: $d = A(:, 3)$

d es la tercera columna de A.



Si digitamos: $C = A\,(([2\ 3]),\ :)$

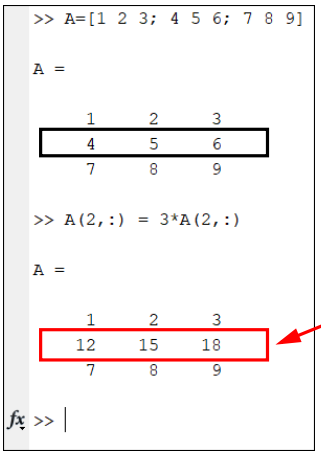
C es la matriz que consiste en el segundo y tercer renglón de A.



Si digitamos: $C = [A\ b]$

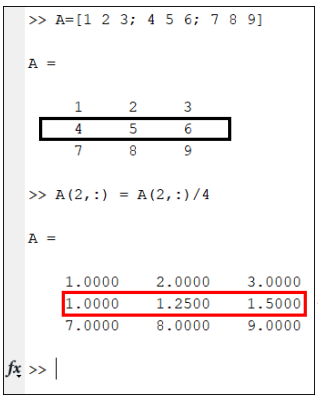
Forma una matriz aumentada $C = (A \mid b)$.

Ahora si deseamos trabajar por medio de la ejecución de operaciones por renglones muy similar a Álgebra Lineal, tenemos las siguientes líneas de programación. Puedes notar que la interpretación de la línea de comandos no difiere en la lógica de escritura de aquello que se desea obtener, especialmente cuando hablamos de vectores y matrices. De igual manera el uso de funciones en MATLAB es muy similar al uso de una calculadora científica o sitio web de cálculo. Te animamos a interiorizar otros ejercicios y funcionalidad básica de MATLAB para tus proyectos siguientes.



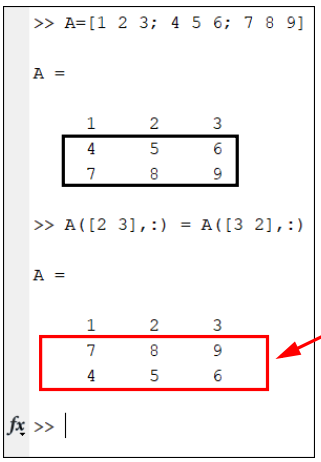
Si digitamos: $A\,(2,:) = 3 * A\,(2,:)$

Equivale a tener $R_2 \to 3R_2$.



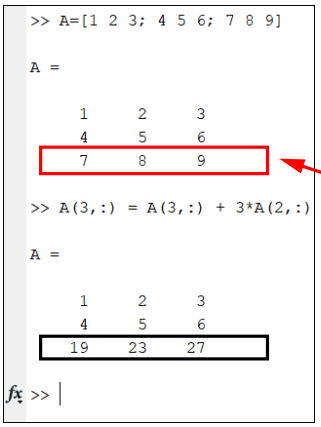
Si digitamos: $A\,(2,:) = A\,(2,:)/4$

Equivale a tener $R_2 \to \frac{1}{4}R_2$.



Si digitamos: $A\,([2\,3],:) = A\,([3\,2],:)$

Intercambia los renglones 2 y 3.



Si digitamos: $A\,(3,:) = A\,(3,:) + 3 * A\,(2,:)$

Equivale a tener $R_3 \to R_3 + 3R_2$.

Un curso gratuito en MathWorks sobre fundamentos de MATLAB puedes encontrar en este enlace https://la.mathworks.com/learn/online-courses/matlab-fundamentals.html.

3.3 Experiencia concreta [Sentir] -Actuar

"*Que el contenido de esta unidad sea el reto final de crear programas que solucionen problemas reales*". Tus profesores.

3.3.1 DESAFÍO

Realice los siguientes ejercicios en base a la experiencia activa:

1. Codifique un programa que realice la conversión de grados $^{\circ}C$ a grados $^{\circ}F$, para ello utilice la fórmula: $^{\circ}F = {}^{\circ}C \times 1.8 + 32$.

Desafío 6. Código en Python del ejercicio1.py

```
# Comentario ejercicio 1 para ayudarte
centigrados = int(input("Ingrese un valor: "))
fahrenheit = (centigrados * 1.8 ) + 32
print("El resultado es: " + str(fahrenheit))  pruebe esta línea
print("El resultado es: " + str(fahrenheit))  pruebe esta línea
```

2. Codifique un programa que realice la conversión de grados $^{\circ}F$ a grados $^{\circ}C$, para ello utilice la fórmula: $^{\circ}C = (^{\circ}F - 32) \div 1.8$
3. Codifique un programa que realice la conversión de grados K a grados $^{\circ}C$, para ello utilice la fórmula: $^{\circ}C = {}^{\circ}K - 273.15$
4. Codifique un programa que realice la conversión de grados $^{\circ}C$ a grados K, para ello utilice la fórmula: $^{\circ}K = {}^{\circ}C + 273.15$
5. Codifique un programa que realice la conversión de grados $^{\circ}F$ a grados K, para ello utilice la fórmula: $^{\circ}K = 5/9\ (^{\circ}F - 32) + 273.15$
6. Codifique un programa que realice la conversión de grados K a grados $^{\circ}F$, para ello utilice la fórmula: $^{\circ}F = 1.8\ (K - 273.15) + 32$
7. Codifique un programa que muestre el área de un cuadrado ingresando el valor de lado.

Desafío 7. Código en Python del ejercicio7.py

```
# Comentario ejercicio 7 para ayudarte
lado = int(input("Ingrese el lado del cuadrado: "))
area = (lado * lado )  puede ser lado**2
print("El área del cuadrado con lado " + str(lado) + " es " , area)
```

8. Codifique un programa que muestre el área de un triángulo ingresando el valor de la base y la altura.
9. Codifique un programa que muestre el área de un rectángulo ingresando el valor de la base y la altura.

10. Codifique un programa que muestre el área de un trapecio ingresando el valor de la base mayor, la base menor y la altura, su fórmula es A= ((B + b) /2) x h.
11. Codifique un programa que muestre el área de un rombo ingresando el valor de la diagonal mayor y la diagonal menor, su fórmula es A= ((D x d) /2).
12. Codifique un programa que permita el ingreso de tres números no iguales e identifique el mayor de ellos.
13. Codifique un programa que permita el ingreso de tres números no iguales e identifique el menor de ellos.
14. Codifique un programa que permita el ingreso de cuatro números no iguales e identifique el mayor de ellos.
15. Codifique un programa que permita el ingreso de cuatro números no iguales e identifique el menor de ellos.
16. Codifique un programa que identifique si un número es Armstrong o no de cuatro cifras. Un número de *n* dígitos se llama Armstrong si la suma de sus dígitos elevados a la *n* es igual al número mismo. Por ejemplo, un número de 4 cifras *abcd* es Armstrong si es igual a $a^4 + b^4 + c^4 + d^4$. Verifique que 9474 es un número de Armstrong y encuentre otro de 4 cifras. Demuestre que no hay números de Armstrong de 2 cifras.
17. Codifique un programa que identifique si un número es Armstrong o no de cinco cifras. Un número de *n* dígitos se llama Armstrong si la suma de sus dígitos elevados a la *n* es igual al número mismo. Por ejemplo, un número de 5 cifras $abcde$ es Armstrong si es igual a $a^5 + b^5 + c^5 + d^5 + e^5$. Verifique que 92727 es un número de Armstrong y encuentre otro de 5 cifras.
18. Codifique un programa que solicite un número y luego muestre en pantalla un mensaje de "Acceso correcto" si el número se encuentra en el rango de números del 70 al 80 (para mostrar el mensaje, el número debe ser mayor e igual a 70 y menor e igual a 80).
19. Codifique un programa que solicite un número y luego muestre en pantalla un mensaje de "Acceso incorrecto" si el número se encuentra en los rangos de números menores a 100 y mayores a 500 (para mostrar el mensaje, el número debe ser menor que 100 y mayor a 500).
20. Codifique un programa que solicite un número y luego muestre en pantalla un mensaje de "Acceso correcto" si el número se encuentra en el rango de números del 18 (sin incluir) al 25 incluido (para mostrar el mensaje, el número debe ser mayor a 18 y menor e igual a 25).

3.4 Reflexión [Analizar] - Observar

"*Que el contenido de esta unidad sea el reto final de crear programas que solucionen problemas reales*". Tus profesores.

3.4.1 EVALUACIÓN

[Lea la página 4 y siga el formato para la entrega de todas las actividades, en un solo documento]

Validación del aprendizaje de la Unidad 3

- Elabora tres funciones en Python que permitan encontrar (a) la potencia de un número como 2^4 y que presente el resultado esperado, para este ejemplo, 16, (b) el factorial de un número, por ejemplo, $4!$, que sería las multiplicaciones de 1x2x3x4 dando como resultado 24 y (c) la secuencia de el enésimo término de la seria de Fibonacci, por ejemplo `fibo(7)` y presente los 7 primeros números de la seria, que son 0 1 1 2 3 5 8.
- Elabora un programa en Python que permita crear 5 arreglos de 50 elementos y llenar de ceros las posiciones pares y de unos las posiciones impares.

Reactivos de la Unidad 3

Pregunta 1. Determine el valor que retorna función:

```
def function ( a, b ):
    if a >b:
        return a
    else:
        return b
# programa principal
x=12
y=13
print (function (x - 2 , y + 2))
```

12	
15	
13	
17	

Pregunta 2. A continuación, se muestra el siguiente código, indique el valor que tiene la variable p, al final de todo el programa.

```
variable = 5
for p in range (variable + 2, 18, 2):
    print (p)
print (p+1)
```

Se obtiene como salida:

13	
17	
15	
18	

Pregunta 3. Sea la siguiente matriz cuadrada de Matlab.

```
P = [2 1 ; 1 4 ]
```

Entonces ¿ la conversión en código en Python de una lista es..?

P = [[2, 1] :[1 , 4]]	
P = [[2, 1] ,[1 , 4]]	
P = [[2 1] [1 4]]	
P = [2 1 ; 1 4]	

Pregunta 4. ¿ Qué hace el siguiente código?

```
numero = 0
while (numero != 7):
    print (“número”)
    numero = numero + 2
```

Entonces la salida es:

Muestra los números pares del 0 al 6	
Muestra el mensaje "número"de manera infinita	
Muestra los números pares de manera infinita	
Muestra el mensaje "número"tres veces	

Finalización de este proceso

El estudio no termina aquí, es constante y dura toda la vida. Aprender nuevas teorías, conceptos, estrategias y buenas prácticas será una actividad en tu día a día.

Esperamos que esta Guía de Aprendizaje, haya sido de utilidad en este primer peldaño de tu sueño de lograr el título de Ingeniera o Ingeniero y por el que nos has dado tu confianza para ser parte de la carrera que elegiste, en nuestra universidad, tu universidad.

Somos nosotros los que te decimos, ¡Gracias!, por tu compromiso de estudiar.

Tu profesor

Bibliografía

[1] J. C. C. McKinsey and G. Boole, "The Mathematical Analysis of Logic.", *Am. Math. Mon.*, vol. 57, no. 5, p. 351, 1950, doi: 10.2307/2306226.

[2] I. Grattan-Guinness, *From the calculus to set theory 1630-1910: An introductory history.* Princeton University Press, 2020.

[3] I. Zubac and S. Rezic, "From Boolean Algebra to PLC Logic", *DAAAM Int. Sci. B.*, pp. 237-244, 2020, doi: 10.2507/daaam.scibook.2020.20.

[4] P. Moretti, "George Boole and Boolean Algebra", 2012.

[5] C. M. C. Mirete, "Álgebras de Boole y la dualidad de Stone", *TEMat Divulg. Trab. Estud. matemáticas*, no. 3, pp. 75-86, 2019.

[6] S. Tymochko et al., "Argumentative Topology: Finding Loop (holes) in Logic", *arXiv Prepr. arXiv* 2011.08952, 2020.

[7] M. Ma and A.-V. Pietarinen, "Peirce's calculi for classical propositional logic", *Rev. Symb. Log.*, vol. 13, no. 3, pp. 509–540, 2020.

[8] T. Hailperin, "Boole's Algebra Isn't Boolean Algebra", *Math. Mag.*, vol. 54, no. 4, pp. 173-184, Sep. 1981, doi: 10.1080/0025570x.1981.11976922.

[9] G. Hitchcock, "New light on George Boole: Desmond MacHale and Yvonne Cohen.", Taylor Francis, 2020.

[10] C. Carlet, "Boolean Functions for Cryptography and Error-Correcting Codes", *Boolean Model. Methods Math. Comput. Sci. Eng.*, pp. 257-397, 2013, doi: 10.1017/cbo9780511780448.011.

[11] J. C. Valverde, H. S. Mortveit, C. Gershenson, and Y. Shi, "Boolean Networks and Their Applications in Science and Engineering", *Complexity*, vol. 2020, 2020.

[12] K. Sarda, A. Yerudkar, and C. Del Vecchio, "Disturbance decoupling control design for Boolean control networks: A Boolean algebra approach", *IET Control Theory Appl.*, vol. 14, no. 16, pp. 2339-2347, 2020, doi: 10.1049/iet-cta.2019.1144.

[13] C. S. Yang and A. Salman Avestimehr, "Coded computing for boolean functions", *arXiv*, 2020.

[14] L. E. J. Brouwer and D. E. Knuth, "Propositional Logic", *Discret. Struct.*, p. 7, 2020.

[15] A. Towler, D. Aranda, R. Ramyaa, and R. Kuo, "Using educational game for engaging students in learning foundational concepts of propositional logic", in *Proceedings-IEEE 20th International Conference on Advanced Learning Technologies, ICALT 2020*, 2020, pp. 208-209, doi: 10.1109/ICALT49669.2020.00067.

[16] K. R. Chowdhary, "Logic and Reasoning Patterns", in *Fundamentals of Artificial Intelligence*, Springer, 2020, pp. 25-50.

[17] P. Smith, *An introduction to formal logic*. Cambridge University Press, 2020.

[18] K. R. Otemaier, E. E. Grein, P. G. Zanese, and N. S. Bosso, "Educational escape room for teaching Mathematical Logic in computer courses".

[19] D. Cerna, M. Seidl, W. Schreiner, W. Windsteiger, and A. Biere, "Aiding an Introduction to Formal Reasoning Within a First-Year Logic Course for CS Majors Using a Mobile Self-Study App", in *Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE, 2020, pp. 61-67, doi: 10.1145/3341525.3387409.

[20] V. Durand-Guerrier, "Logic in Mathematics Education", *Encycl. Math. Educ.*, pp. 361-364, 2014, doi: 10.1007/978-94-007-4978-8_92.

[21] M. Trigueros and M. Wawro, "Linear Algebra Teaching and Learning", *Encycl. Math. Educ.*, pp. 1-5, 2019, doi: 10.1007/978-3-319-77487-9_100021-1.

[22] V. J. Rayward-Smith, T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, vol. 42, no. 9. MIT press Cambridge, 1991.

[23] S. S. Skiena, "Introduction to Algorithm Design", 2020, pp. 3-30.

[24] N. D. C. Alves, C. G. Von Wangenheim, J. C. R. Hauck, and A. F. Borgatto, "A large-scale evaluation of a rubric for the automatic assessment of algorithms and programming concepts", in *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 2020, pp. 556-562, doi: 10.1145/3328778.3366840.

[25] N. da Cruz Alves, C. G. Von Wangenheim, J. C. R. Hauck, A. F. Borgatto, and D. F. de Andrade, "An Item Response Theory Analysis of the Sequencing of Algorithms & Programming Concepts", *Cool*. JC, p. 9, 2020.

[26] L. Joyanes Aguilar, "Fundamentos de programación: algoritmos y estructura de datos y objetos", 2003.

[27] S. S. Skiena, "Algorithm Analysis", 2020, pp. 31-67.

[28] N. Kumar, S. Aggarwal, and P. R. Chelliah, "Data structures", in *Advances in Computers*, 2020, pp. 69-108.

[29] M. Kantaria, G. Basilaia, M. Dgebuadze, and G. Chokhonelidze, "Applying a new teaching methodology to university programming language courses", *Int. J. Educ. Res.*, vol. 8, no. 4, 2020, [Online]. Available: www.ijern.com.

[30] E. Seralidou and C. Douligeris, "Learning programming by creating games through the use of structured activities in secondary education in Greece", *Educ. Inf. Technol.*, pp. 1-40, 2020, doi: 10.1007/s10639-020-10255-8.

[31] A. B. Chaudhuri, *Flowchart and Algorithm Basics: The Art of Programming*. Stylus Publishing, LLC, 2020.

[32] G. Gan, C. Ma, and J. Wu, *Data clustering: theory, algorithms, and applications*. SIAM, 2020.

[33] A. Fallis, *Essential Algorithms: A Practical Approach to Computer Algorithms*, vol. 53, no. 9. John Wiley Sons, 2013.

[34] M. A. Carreño-León, F. J. Rodríguez-Álvarez, and J. A. Sandoval-Bringas, "Using gamification technique to increase capacity in the resolution of problems during the process teaching and learning programming", in *Advances in Intelligent Systems and Computing*, 2019, vol. 909, pp. 195-203, doi: 10.1007/978-3-030-11434-3_23.

[35] J. Llerena-Izquierdo and J. Idrovo-Llaguno, "Introducing Gamification to Improve the Evaluation Process of Programing Courses at the Salesian Polytechnic University (Guayaquil, Ecuador)", *Adv. Intell. Syst. Comput.*, vol. 1273 AISC, pp. 402-412, 2021, doi: 10.1007/978-3-030-59194-6_33.

[36] J. Llerena-Izquierdo and L. Atiaja-Balseca, "Gamification Within the Learning Evaluation Process Using Ardora at the Salesian Polytechnic University (Guayaquil, Ecuador)", pp. 139-150, Dec. 2021, doi: 10.1007/978-3-030-71503-8_11.

[37] J. Llerena-Izquierdo and J. Zamora-Galindo, "Using H5P Services to Enhance the Student Evaluation Process in Programming Courses at the Universidad Politécnica Salesiana (Guayaquil, Ecuador)", pp. 216-227, Oct. 2021, doi: 10.1007/978-3-030-68080-0_16.

[38] S. S. Skiena, *The algorithm design manual: Second edition*. Springer International Publishing, 2008.

[39] R. Stephens, *Essential Algorithms: A Practical Approach to Computer Algorithms Using Python and C*. John Wiley Sons, 2019.

[40] J. A. Martínez-Valdés, F. J. García-Pẽalvo, and J. A. Velázquez-Iturbide, "The role of basic mathematics concepts in programming teaching and learning", in *ACM International Conference Proceeding Series*, 2019, pp. 1046–1054, doi: 10.1145/3362789.3362933.

[41] A. Kumar Bansal, *Introduction to programming languages*. CRC Press, 2013.

[42] M. Ortiz and A. Plaza, "Programación orientada a objetos con Java y UML", Oct-2013. https://dspace.ups.edu.ec/handle/123456789/18054 (accessed Dec. 25, 2020).

[43] J. Llerena Izquierdo, "Codifica en Python." http://dspace.ups.edu.ec/handle/123456789/19346.

[44] C. Wang, "A Brief Introduction of Python to Freshman Engineering Students Using Multimedia Applications", in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1-8.

[45] J. Wang, L. Li, K. Liu, and H. Cai, "Exploring how deprecated Python library APIs are (not) handled", in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 233-244, doi: 10.1145/3368089.3409735.

[46] M. Wende, T. Giese, S. Bulut, and R. Anderl, "Framework of an active learning python curriculum for first year mechanical engineering students", in *IEEE Global Engineering Education Conference, EDUCON*, 2020, vol. 2020-April, pp. 1193-1200, doi: 10.1109/EDUCON45650.2020.9125259.

[47] N. A. Azzam and R. A. A. Al-Kayyali, “An Integrated Method for STEM Education using MATLAB”, in *2020 Advances in Science and Engineering Technology International Conferences, ASET 2020*, 2020, pp. 1-4, doi: 10.1109/ASET48392.2020.9118265.

[48] M. M. Rahman, M. H. Sharker, and R. Paudel, “Active and Collaborative Learning Based Dynamic Instructional Approach in Teaching Introductory Computer Science Course with Python Programming”, in *2020 IEEE Integrated STEM Education Conference (ISEC)*, 2020, pp. 1-7.

[49] A. Rawat, “A Review on Python Programming”, *Int. J. Res. Eng. Sci. Manag.*, vol. 3, no. 12, pp. 8-11, 2020.

[50] N. Silaparasetty and N. Silaparasetty, “Python Programming in Jupyter Notebook“, in *Machine Learning Concepts with Python and the Jupyter Notebook Environment*, Springer, 2020, pp. 119-145.

[51] R. Hosseini et al., “Improving Engagement in Program Construction Examples for Learning Python Programming”, *Int. J. Artif. Intell. Educ.*, vol. 30, no. 2, pp. 299-336, 2020, doi: 10.1007/s40593-020-00197-0.

[52] C. E. Widodo, K. Adi, and R. Gernowo, “Medical image processing using python and open cv”, in *Journal of Physics: Conference Series*, 2020, vol. 1524, no. 1, p. 12003, doi: 10.1088/1742-6596/1524/1/012003.

[53] L. Gojković, S. Malijević, and S. Armaković, “Modeling of fundamental electronic circuits by the Euler method using the Python programming language”, *Phys. Educ.*, vol. 55, no. 5, p. 55016, 2020, doi: 10.1088/1361-6552/ab94d5.

[54] J. Tang, “Introduction to Python programming in finance”, 2020.

[55] J. Xu and M. Frydenberg, “Python Programming in an IS Curriculum: Perceived and Outcomes”, in *Communications of the Association for Information Systems*, 2019, vol. 44, no. January, pp. 123-155.

[56] T. ElAli, “Discrete Signals and Systems with MATLAB”, *Discret. Signals Syst. with MATLAB*, 2020, doi: 10.1201/9781003088592.

[57] P. P. Cruz, A. M. Gutiérrez, R. A. Ramírez-Mendoza, E. M. Flores, A. A. Ortiz Espinoza, and D. C. Balderas Silva, *A Practical Approach to Metaheuristics Using LabVIEW and MATLAB*. CRC Press, 2020.

[58] X. Yi and H. Chune, “Research on Flipped Classroom Teaching of Communication Principle Experiment on Micro Course and MATLAB”, *Sci. J. Educ.*, vol. 8, no. 2, p. 47, 2020, doi: 10.11648/j.sjedu.20200802.13.

Carrera de Computación

El objetivo de la *Guía de aprendizaje de Programación* es enseñar programación a jóvenes en sus primeros años de estudios universitarios. Ofrece contenidos introductorios en la programación básica. El texto se desarrolla en tres unidades: Lógica Matemática, Algoritmos y la tercera a los Lenguajes de Programación, específicamente Python y MatLab.

Este volumen plantea ejercicios desarrollados y resolución de problemas, mediante el análisis, diseño y desarrollo de soluciones utilizando lógica algorítmica. Además, sigue el currículo diseñado por profesores que pertenecen al Claustro de Computación Aplicada y Metodologías de la Computación (CAMC) de la sede Guayaquil de la Universidad Politécnica Salesiana, quienes motivados por el progreso en el desempeño académico de sus estudiantes, han seleccionado problemas y ejercicios significativos para el aprendizaje de la programación; modelos sencillos para resolver con orientaciones para encontrar alternativas de solución.

ABYA YALA | UPS

ECUADOR SALESIANOS DON BOSCO

ISBN-13: 978-9978-10-769-0

9 789978 107690