

ESTUDIO COMPARATIVO ENTRE PSEINT Y RAPTOR BASADO EN LA ESTRUCTURA WHILE

AUTORES

Grace Liliana Figueroa Morán
Julio Pedro Paladines Morán
José Nevardo Paladines Morán
Lilian de Jesús Aguilar Ponce
Marjorie Marina Moreira Mero
Yanina Holanda Campozano Pilay
Edwin Joao Merchán Carreño





Primera Edición 2023

ISBN: 978-9942-7090-6-6

2023, ALEMA Casa Editora-Editorial Internacional S.A.S.D

Jipijapa – Manabí – Ecuador

<https://editorialalema.org/libros/index.php/alema>

Diseño y diagramación:

Ing. Xiomara Lisbeth Anzules Avila

Corrección de contenidos:

Ing. Wilter Leonel Solórzano Álava, Mg.

Diseño, montaje y producción editorial:

ALEMA Casa Editora-Editorial Internacional S.A.S.D, Ecuador

Hecho en Ecuador

Made in Ecuador

Este texto ha sido sometido a un proceso de evaluación por pares externos.

Advertencia: “Quedan todos los derechos reservados. Se prohíbe la reproducción, el registro o la transmisión parcial o total de esta obra por cualquier sistema de recuperación de información existente o por existir, sin el permiso previo por escrito del titular de los derechos correspondientes”.

ISBN: 978-9942-7090-6-6



LIBRO: ESTUDIO COMPARATIVO ENTRE PSEINT Y RAPTOR BASADO EN LA ESTRUCTURA WHILE

**Dedicado para aquellas personas que les guste la tecnología y quieran aprender más sobre
el uso de ciertas aplicaciones.**

“Cualquier tecnología suficientemente avanzada es equivalente a la magia.”

(Arthur C. Clarke)

*“La sociedad tecnológica ha logrado multiplicar las ocasiones de placer, pero encuentra muy difícil
engendrar la alegría”.*

(Papa Francisco)

AUTORES



Grace Liliana Figueroa Morán

Doctora en Tecnologías de la Información y Comunicaciones.

Docente de la carrera Tecnología de la Información
Universidad Estatal del Sur de Manabí. Ecuador.

grace.figueroa@unesum.edu.ec

<https://orcid.org/0000-0003-2520-765X>



Julio Pedro Paladines Morán

Magíster en Informática Empresarial.

Docente de la carrera Tecnología de la Información
Universidad Estatal del Sur de Manabí. Ecuador.

julio.paladines@unesum.edu.ec

<https://orcid.org/0000-0002-8121-3360>



José Nevardo Paladines Morán

Doctor en Software, Sistemas y Computación,

Docente de la carrera Tecnología de la Información
Universidad Estatal del Sur de Manabí. Ecuador.

jose.paladines@unesum.edu.ec

<https://orcid.org/0000-0003-1991-1894>



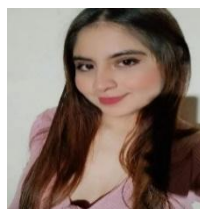
Lilian de Jesús Aguilar Ponce

Magíster en Auditoría Integral

Magíster en Educación, Mención educación y
creatividad. Docente de la Unidad Educativa Quince
de Octubre. Ecuador.

lilijesusodani@gmail.com

<https://orcid.org/0000-0002-6365-5130>



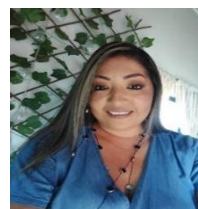
Marjorie Marina Moreira Mero

Estudiante de Tecnología de la Información

Universidad Estatal del Sur de Manabí. Ecuador.

moreira-marjorie9426@unesum.edu.ec

<https://Orcid:0000-0001-6871-1106>



Yanina Holanda Campozano Pilay

Magíster en Gerencia Educativa.

Docente de la carrera Tecnología de la Información
Universidad Estatal del Sur de Manabí. Ecuador.

holanda.campozano@unesum.edu.ec

<https://orcid.org/0000-0001-5319-6076>



Edwin Joao Merchán Carreño

Doctor en Tecnologías de la Información y Comunicaciones

Docente de la carrera Tecnología de la Información

Universidad Estatal del Sur de Manabí. Ecuador.

joao.merchan@unesum.edu.ec

<https://orcid.org/0000-0001-8128-2764>

Índice de contenidos

PRÓLOGO.....	viii
RESUMEN	ix
ABSTRACT.....	x
CAPÍTULO I. INTRODUCCIÓN	1
CAPÍTULO II. MARCO TEÓRICO	6
2.1 PSeInt	6
2.2.1. Historia y creación de PSeInt.....	6
2.2.2. Objetivos de PSeInt.....	7
2.2.3. Características y funcionalidades de PSeInt:	7
2.2.4. Interfaz de trabajo	8
2.2.5. PSeInt: Acciones (Comandos):	9
2.2.6. Estructura de PSeInt.....	9
2.2.7. Flujogramas y ejercicios	22
2.2.8. Funciones / Subprocesos en PSeInt	23
2.2 RAPTOR.....	24
2.2.1. Funcionamiento del programa RAPTOR.....	24
2.2.2. ¿Por qué utilizar RAPTOR?.....	25
2.2.3. Estructura del programa RAPTOR	25
2.2.4. Características del programa RAPTOR	27
2.2.5. Interfaz de trabajo	28
2.3. Modelos de programación.....	30
2.3.1. Modelo declarativo	30
2.3.2. Modelo imperativo	30
2.3.3. Modelo Orientado a Objetos.....	31
2.4. Técnicas de programación.....	31
2.4.1. Programación estructurada.....	32
2.4.2. Programación modular.....	34
2.4.3. Programación Orientada a Objetos (POO)	34
2.4.4. Programación concurrente	34
2.4.5. Programación lógica	34

2.5. ¿En qué profesiones hace falta saber programar?	35
CAPÍTULO III. METODOLOGÍA	36
3.1. Tipo de investigación	36
3.2. Métodos	36
3.2.1. Método Analítico-Sintético.....	36
3.2.2. Método Bibliográfico.....	36
3.3. Materiales	36
CAPÍTULO IV. RESULTADOS.....	37
4.1. Descripción de los resultados	37
4.2. Resolución de ejercicios	37
4.3. Análisis de resultados	44
CAPÍTULO V. CONCLUSIONES	47
RECOMENDACIONES.....	48
REFERENCIAS BIBLIOGRÁFICAS.....	49

Índice de figuras y tablas

Figura 1.- Programación en PSeInt.	8
Figura 2.- Diagrama de Flujo en PSeInt	22
Figura 3.- Divisores exactos en PSeInt.	37
Figura 4.- Diseño en Raptor.....	38
Figura 5.- Diferencias de números positivos	38
Figura 6.- Diagrama de flujo de números positivos.....	39
Figura 7.- Programa de N números primos.....	39
Figura 8.- Diagrama de Flujo de N números primos	40
Figura 9.- N salarios en PSeInt	41
Figura 10.- Diagrama de Flujo de N salarios	42
Figura 11.- Diagrama N salarios en PSeInt	43
Tabla 1.- Análisis comparativo PSeInt y RAPTOR	44

PRÓLOGO

La programación es una habilidad cada vez más importante en el mercado laboral actual. Con la creciente demanda de profesionales en tecnología de la información, es esencial que los estudiantes tengan acceso a herramientas y métodos de enseñanza efectivos para que puedan adquirir estas habilidades de manera adecuada.

En este libro, se presenta un estudio comparativo entre dos herramientas populares de enseñanza de programación: PSeInt y RAPTOR. Estas herramientas se utilizan comúnmente para enseñar programación a principiantes, y se enfocan en la enseñanza de la estructura “*while*”.

A través de este estudio comparativo, se pretende determinar cuál de estas herramientas es más efectiva en la enseñanza de la estructura “*while*”. Al analizar las similitudes y diferencias entre PSeInt y RAPTOR, se espera proporcionar información útil para educadores, estudiantes y programadores interesados en mejorar su comprensión de la programación.

Este libro no solo es importante para aquellos que buscan mejorar sus habilidades de programación, sino que también puede ser útil para aquellos que buscan una introducción a la programación. Además, puede ser una valiosa herramienta para educadores que desean mejorar su enseñanza de la programación.

Espero que este libro sea útil para aquellos que buscan mejorar sus habilidades de programación y para aquellos que buscan una introducción a este importante campo de estudio.

¡Disfruten de la lectura!

RESUMEN

En la actualidad, el uso de los programas representa un rol muy importante tanto en la carrera de ingeniería, como en las tecnologías. Es imprescindible el uso de estos programas porque resulta muy satisfactorio para muchos estudiantes, al momento de representar gráficamente soluciones de problemas, para obtener el código de un programa de computadora, y al realizar una gran cantidad de ejercicios y problemas con métodos a codificar, probar y corregir hasta llegar a la solución más óptima.

El texto que se presenta tiene como objetivo realizar un análisis comparativo de las herramientas PSeInt y RAPTOR que son programas utilizados para elaborar algoritmos en su estructura de repetición *While*.

PSeInt y RAPTOR son herramientas que simplifican y facilitan el trabajo en el desarrollo de soluciones. Por un lado, PSeInt es un lenguaje de programación que se basa en pseudocódigo. Esta herramienta proporciona facilidades para la escritura de los algoritmos e implementa algunas herramientas para la detención de errores, también la comprensión de la lógica algorítmica.

Por otro lado, RAPTOR es un programa muy útil para crear diagramas de flujo. Es muy necesaria la solución de ciertos errores que lleva a cabo la verificación de problemas, así como en el ejercicio haciendo uso de términos variables, nombre, números, entre otros.

En resumen, PSeInt se basa en encontrar fallos y errores en un algoritmo, ya que es una herramienta simple e intuitiva de pseudocódigo y RAPTOR es un software libre, un programa en el cual nos señala de forma específica dónde tenemos el error.

Para lograr el análisis comparativo entre las herramientas de estudio, hemos propuesto cuatro ejercicios que se han realizado tanto en PSeInt como en RAPTOR, con ello, hemos podido identificar algunos aspectos propios de cada herramienta, con las que fue posible fijar algunas similitudes y diferencias entre ellas.

ABSTRACT

At present, the use of the programs represents a very important role both in the engineering career and in the technologies. The use of these programs is essential because it is very satisfactory for many students, when graphically representing solutions to problems, to obtain the code of a computer program, and when carrying out a large number of exercises and problems with methods to be coded, tested and correct until reaching the most optimal solution.

The objective of the text presented is to carry out a comparative analysis of the PSeInt and RAPTOR tools, which are programs used to develop algorithms in their While repetition structure.

PSeInt and RAPTOR are tools that simplify and facilitate the work in developing solutions. On the one hand, PSeInt is a programming language that is based on pseudocode. This tool provides facilities for writing algorithms and implements some tools for error detection, as well as understanding algorithmic logic.

On the other hand, RAPTOR is a very useful program for creating flowcharts. It is very necessary to solve certain errors that carry out the verification of problems, as well as in the exercise making use of variable terms, name, numbers, among others.

In summary, PSeInt is based on finding flaws and errors in an algorithm, since it is a simple and intuitive pseudocode tool and RAPTOR is free software, a program in which it specifically indicates where we have the error.

To achieve the comparative analysis between the study tools, we have proposed four exercises that have been carried out in both PSeInt and RAPTOR, with this, we have been able to identify some aspects of each tool, with which it was possible to establish some similarities and differences between them.

CAPÍTULO I. INTRODUCCIÓN

Uno de los problemas a los que se enfrenta un estudiante que da sus primeros pasos en el aprendizaje de la programación reside en que las palabras reservadas de la mayoría de los lenguajes se basan en el inglés, por lo que al esfuerzo de añadir un nuevo lenguaje informático se añade la tarea de estar traduciendo mentalmente términos como “*printf*”, “*readln*” o “*assertEqual*”.

Para evitar este problema, y para asegurarse de que esas primeras nociones sobre programación no sean aplicables a un único lenguaje, se suele recurrir a lo que llamamos “pseudocódigo”: un falso lenguaje que represente en la solución a un algoritmo de la forma más comprensible posible, manteniendo una estructura similar al código real, y todo ello en la lengua en la que “piensa” el estudiante.

Pseudocódigo: Descripción detallada y legible de lo que debe hacer un programa de computadora o un algoritmo, expresado en un lenguaje natural de estilo formal en lugar de un lenguaje de programación. Se utiliza como un paso detallado en el proceso de desarrollo de un programa. El pseudocódigo proporciona a los programadores una plantilla detallada para el siguiente paso de escribir código en un lenguaje de programación específico.

“The algorithmic analysis of hybrid systems”

(Alur et al., 1995)

El pseudocódigo es un recurso valioso que se utiliza en la educación en programación, el desarrollo de software y los informes científicos para diseñar soluciones algorítmicas, ya que es fácil de escribir, comprender y modificar (Del Prado & Lamas, 2014). Dado que el pseudocódigo carece de la capacidad de ser probado, es difícil determinar si una solución de pseudocódigo es correcta o no. Las herramientas de software son especialmente necesarias para alcanzar este objetivo, por ejemplo, ayudar a los profesores a encontrar condiciones de carrera, interbloqueos o problemas de inanición mientras califican el pseudocódigo concurrente de los estudiantes (Ulate-Caballero et al., 2021).

Muchas veces, el objetivo de este pseudocódigo es el de ser usado en una pizarra o en un ejercicio con lápiz y papel, antes de empezar a usar entornos de programación reales con lenguajes también reales (Verner, I. M., & Card, D. N., 2018). En ese sentido, este texto integrador de saberes está dirigido a revisar minuciosamente las bondades de dos herramientas muy prácticas y de actual utilidad como son: PSeInt y RAPTOR. A partir de estas herramientas pretendemos estudiarlas y realizar ejercicios que nos permitan establecer comparaciones entre ellas.

La programación es una habilidad esencial en la era digital en la que vivimos, y hay muchas herramientas y lenguajes de programación disponibles para aprender y enseñar la lógica de programación. Dos de las herramientas de programación visual más populares son PSeInt y RAPTOR, que se utilizan comúnmente para enseñar la estructura de control de bucles *While* (Fernández-Sanz, L., & Escalona, M. J., 2021).

RAPTOR, creado por *Air Force Academy* en 2005, utiliza un entorno de modelo de flujo para enseñar conceptos de programación (Cooper, 2014). Es un IDE diseñado para la enseñanza de la programación que hace uso de diagramas de flujo como forma de escribir algoritmos. Los diagramas de flujo escritos en Raptor tienen los símbolos típicos de los diagramas de flujo: lectura, escritura, procesamiento y decisión, agregando un nuevo símbolo para editar variables. Permite la implementación completa del diagrama de flujo y la ejecución paso a paso, lo que permite recuperar los valores almacenados para cada variable (Manso et al., 2019).

Pseint fue desarrollado por Pablo Novara en 2004, como herramienta para la enseñanza de la lógica de programación a través de un pseudocódigo muy potente, que mantiene las mismas herramientas de un lenguaje de alto nivel y con instrucciones en español, lo que permite al alumno enfocarse directamente en la lógica de la programación, sin tener que conocer toda la codificación técnica en inglés (Castrillón et al., 2022).

Este libro se centra en un estudio comparativo entre PSeInt y RAPTOR basado en la estructura “*while*”, para determinar cuál de estas herramientas es más efectiva para enseñar y comprender esta estructura. A través de este estudio, se analizarán aspectos como la capacidad de visualización, la facilidad de uso, las funcionalidades, la comunidad y los recursos disponibles, así como la perspectiva de futuro.

Este libro está dirigido a estudiantes y profesores de programación, así como a cualquier persona interesada en aprender más sobre estas herramientas y su efectividad en la enseñanza de la

programación. Se espera que este estudio comparativo sea de gran ayuda para los principiantes en programación que buscan una herramienta visual efectiva para aprender la estructura de control de bucles *While*.

El Libro está conformado por cinco capítulos, en el que se integran un conjunto de saberes esenciales para la introducción a la programación:

En el primer capítulo presentamos la Introducción donde se describe todo el contenido que se abordará en el libro. Además, se describe la problematización que contiene la contextualización del problema donde refleja la poca utilidad que se le da en la utilización de las herramientas PSeInt y RAPTOR para construir algoritmos, siguiendo con la formulación del problema central, y los objetivos: general y específicos.

El segundo capítulo Marco Teórico, está puntualizado la fundamentación teórica de las variables del objeto de estudio, haciendo énfasis de los contenidos tales como: PSeInt, RAPTOR y técnicas de programación. El objetivo principal de este capítulo es capacitar al estudiante en los conceptos básicos de algoritmos, que le permitan obtener la destreza necesaria para diseñar sus propios algoritmos.

El tercer capítulo, Metodología, se presentan los métodos y técnicas que utilizamos en el proceso investigativo y desarrollo de los algoritmos.

El cuarto capítulo, Resultados, registra los resultados del análisis que contiene el código fuente de los programas que se han realizado en las herramientas PSeInt y RAPTOR y un comentario final de la comparación de estos.

En el capítulo cinco se presentan las conclusiones derivadas del proceso de investigación y análisis sobre las herramientas PSeInt y RAPTOR.

Al final incluimos un conjunto de recomendaciones, y la bibliografía que sustenta la investigación, y que constituye una base de consulta para profundizar los saberes.

Problemática

En la asignatura de Fundamentos de Programación se trata el contenido relacionado con el desarrollo de algoritmos, sin embargo, existe una poca utilidad en la utilización de las herramientas PSeInt y RAPTOR para construir algoritmos.

Problema

No existe un análisis comparativo de las aplicaciones PSeInt y RAPTOR al momento de implementar soluciones informáticas y de forma específica al momento de utilizar la estructura de repetición *While*.

Objetivos

Objetivo General

Realizar un análisis comparativo de PSeInt y RAPTOR en la utilización de la estructura de repetición *While*.

Objetivos específicos

- Estudiar conceptualmente las aplicaciones PSeInt y RAPTOR como herramientas para representar gráficamente soluciones de problemas y obtener el código de un programa de computadora.
- Revisar cómo se implementan las técnicas de programación en las aplicaciones PSeInt y RAPTOR.
- Proponer ejemplos de aplicación en PSeInt y RAPTOR en los que utilicen la estructura de repetición *While*, con los que se pueden determinar diferencias y semejanzas entre las aplicaciones.

Importancia del libro

La importancia de este libro radica en que se está abordando de forma detallada y con ejercicios demostrativos, un tema relevante en la educación y en la programación. En la actualidad, existen diversas herramientas y lenguajes de programación que se utilizan para enseñar a los estudiantes cómo programar. PSeInt y RAPTOR son dos de estas herramientas que se utilizan comúnmente para enseñar programación a principiantes.

Al realizar un estudio comparativo entre estas dos herramientas, se está contribuyendo al conocimiento existente sobre la efectividad de diferentes herramientas en la enseñanza de la programación, específicamente en lo que se refiere a la estructura *while*. Los resultados registrados en este libro tienen gran utilidad para educadores, estudiantes y programadores interesados en enseñar o aprender programación de manera más efectiva.

Además, la investigación en la enseñanza de la programación es importante porque la programación se está convirtiendo en una habilidad cada vez más importante en el mercado laboral actual (Rodríguez et al., 2021). Por lo tanto, es esencial que los estudiantes tengan acceso a herramientas y métodos de enseñanza efectivos para que puedan adquirir estas habilidades de manera adecuada y estar preparados para enfrentar los desafíos del mercado laboral actual y futuro.

En resumen, este libro es muy importante porque contribuye al conocimiento existente sobre la enseñanza de la programación, específicamente en lo que se refiere a PSeInt y RAPTOR y su efectividad en la enseñanza de la estructura *while*.

CAPÍTULO II. MARCO TEÓRICO

2.1 PSeInt

PSeInt (Pseudo Intérprete), es un Entorno de Desarrollo Integrado (IDE) para pseudocódigo, un lenguaje de programación imperativa simple y en castellano. Es decir, PSeInt es un editor e intérprete de programas escritos en pseudocódigo. Su interfaz gráfica permite crear, almacenar, ejecutar y corregir fácilmente programas en pseudocódigo. La sencillez del lenguaje pseudocódigo lo hacen ideal para la enseñanza de la programación (Laura-Ochoa & Bedregal-Alpaca, 2022).

Mediante el uso de un lenguaje de pseudocódigo simple y limitado y una interfaz de usuario intuitiva en español. Los estudiantes pueden comenzar a comprender conceptos básicos de estructuras de control, apoyados en un paradigma de programación modular y procedimental. Además, esta herramienta permite el uso de diagramas de flujo, funciones y procedimientos con parámetros, y también se admiten arreglos multidimensionales (Castillo-Barrera et al., 2013).

PSeInt Permite escribir programas con instrucciones condicionales (*Si-Entonces-Sino, Según*) y ciclos (*Mientras, Hasta Que, Para*), y también usar valores numéricos (números decimales), lógicos, caracteres y arreglos. También provee funciones de entrada/salida y algunas funciones matemáticas.

2.2.1. Historia y creación de PSeInt

PSeInt es un intérprete de pseudocódigo completamente en español. El software comenzó a desarrollarse en octubre de 2003 utilizando Borland C++ Builder debido a que era esta la herramienta que recomendaba la cátedra por la facilidad que brinda para el desarrollo de interfaces visuales (Manso et al., 2019). Se comenzó con una prueba de concepto, ya que era el primer proyecto que iba a desarrollar con C++. A esta inexperiencia con el lenguaje, los creadores de la herramienta le atribuyen errores de diseño y muchas ineficiencias en la implementación que con el tiempo fueron corrigiendo.

Al ser este un software para uso exclusivamente didáctico, no se presentan grandes problemas de rendimiento, puesto que no se tienen que interpretar algoritmos de miles de líneas sumamente complejos (Álava et al., 2022a, 2022b).

PSeInt: es un entorno de programación diseñado para ayudar a los estudiantes a aprender programación. Es gratuito, de código abierto y está disponible para Windows, Mac y Linux. PSeInt utiliza un lenguaje de programación similar a Pseudocódigo y se enfoca en enseñar a los estudiantes a resolver problemas mediante la programación.

A tool for teaching computational thinking

(Agarwal et al., 2018)

2.2.2. Objetivos de PSeInt

Muchas veces una de las cosas que se les hace más difícil a los nuevos estudiantes de informática es aprender la sintaxis, ya que generalmente aparece en idioma inglés. Basado en este principio los desarrolladores de sistemas computacionales comenzaron a cuestionarse por que no existía algo que en vez de utilizar “for” usara “para”, o en lugar de “if” usase “si”. Estas intenciones iniciales ya hoy sí existen.

PSeInt está pensado para asistir a los estudiantes que se inician en la construcción de programas o algoritmos computacionales. El pseudocódigo se suele utilizar como primer contacto para introducir conceptos básicos como el uso de estructuras de control, expresiones, variables, etc., sin tener que lidiar con las particularidades de la sintaxis de un lenguaje real (Ulate-Caballero et al., 2021).

Este software pretende facilitarle al principiante la tarea de escribir algoritmos en este pseudolenguaje presentando un conjunto de ayudas y asistencias, y brindarle además algunas herramientas adicionales que le ayuden a encontrar errores y comprender la lógica de los algoritmos.

El objetivo de PSeInt es introducir al mundo de la programación de manera sencilla y lo mejor de todo es ¡Software Libre! con licencia GPL.

2.2.3. Características y funcionalidades de PSeInt:

En el texto de lectura recomendada: “*PSeInt as a learning tool for programming fundamentals*”, se han descrito las siguientes características y funcionalidades de PSeInt (Pérez et al., 2018):

Funcionalidades de PSeInt

- ❖ Presenta herramientas de edición básicas para escribir algoritmos en PseudoCódigo en español
- ❖ Permite la edición simultánea de múltiples algoritmos
- ❖ Presenta ayudas para la escritura
- ❖ Autocompletado
- ❖ Ayudas Emergentes
- ❖ Plantillas de Comandos
- ❖ Coloreado de Sintaxis
- ❖ Indentado Inteligente
- ❖ Puede ejecutar los algoritmos escritos.

La figura 1, muestra el entorno de trabajo de PSeInt.

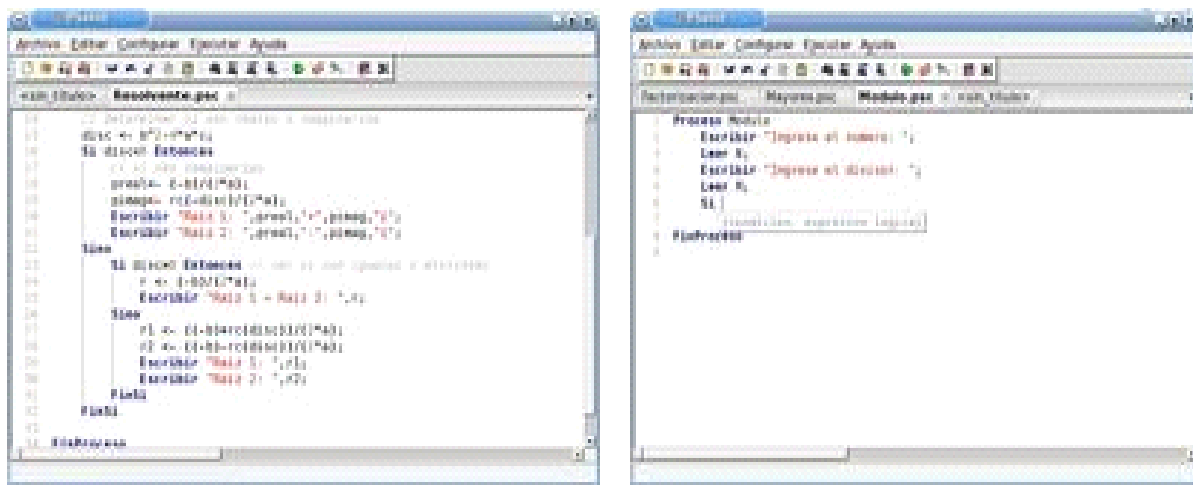


Figura 1.- Programación en PSeInt.

2.2.4. Interfaz de trabajo

Basado en la opinión de los propios desarrolladores, la herramienta PSeInt tiene los siguientes inconvenientes (Viana, J. P. S., & Pilla, A. M., 2021):

- Se carece de soporte para el análisis del problema y planteamiento de la solución, aun cuando permite editar comentarios inmersos en el pseudocódigo.
- La ejecución simula a la mayoría de los ambientes de desarrollo y no visualiza el cambio de valor de las variables.
- No presenta la traza completa de la ejecución.

- El diagrama de flujo generado no concuerda del todo con la notación estándar de la mayoría de los libros de algoritmos en lo que respecta a entrada, salida y el ciclo “para”.
- No se realizan validaciones semánticas para evitar ciclos infinitos

2.2.5. PSeInt: Acciones (Comandos):

Un comando es una orden que se le da a un programa de computadora que actúa como intérprete del mismo, para así realizar una tarea específica.

Los Comandos son utilizados para generar acciones dentro del programa. Estos pueden variar desde Escribir o Leer hasta hacer funciones matemáticas y más.

Propiedades principales:

Los Comandos mencionados aquí deben estar dentro del Algoritmo principal, es decir:

Ninguna acción/comando debe tener comillas de ningún tipo en el nombre del comando, ejemplo:

Correcto:	Incorrecto:
<ul style="list-style-type: none"> • Escribir • Leer 	<ul style="list-style-type: none"> • ‘Escribir’ • ‘Leer’

2.2.6. Estructura de PSeInt

La estructura de un algoritmo sirve para organizar a los elementos que aparecen en él. Todos los algoritmos tienen la misma estructura:

- **Cabecera:** En la cabecera de un algoritmo se debe de indicar el nombre (identificador) asignado al mismo.

`Algoritmo sin_titulo`

`FinAlgoritmo`

- **Dimensionamiento:** Esta instrucción define un arreglo con el nombre indicado en <identificador> y N dimensiones.

- **Declaración de variables:** En esta sección se declaran las constantes, los tipos de datos y las variables que se usan en un algoritmo.
- **Entrada de datos:** Aquí se ingresan los datos que se van a procesar.
- **Proceso:** Se analizan y ordenan los datos para realizar las acciones que el pseudocódigo requiere.
- **Escribir:** Se utiliza para mostrar un texto en el programa.

PSeInt: Comando de Escribir:



Hay dos tipos de Escribir:

- Escribir: Es normal, pero al terminar el texto, enviará a la línea de abajo.
- Escribir Sin Saltar: Es utilizado para mostrar un texto sin enviar a la línea de abajo.

Desarrollo:

Si es texto (no variable asignada) debe estar encerrado entre comillas ya sean simples o dobles (Propiedades de los datos), ejemplo: 'Texto'

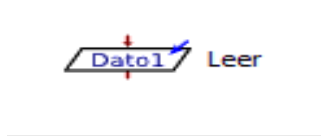
Ejemplo en programa:

```
Escribir 'Hola'
Escribir Sin Saltar '¿Como '
Escribir 'estas?'
```

```
*** Ejecución Iniciada. ***
Hola
¿Como estas?
*** Ejecución Finalizada. ***
```

Como puede observar, el 'Escribir Sin Saltar'

PSeInt: Comando Leer



Leer: Permite al usuario asignar el valor a una variable, ya sea Numérico o Carácter. Desarrollo:

- Debes indicar el nombre de una variable
- Esta puede ser usada en 'Escribir' siempre y cuando no utilice comillas de ningún tipo
- La acción 'Leer' debe estar previa a la variable, ejemplo: Leer nombre Variable

```
Escribir '¿Como te llamas?'  
Leer nombrePersona  
Escribir Sin Saltar 'Hola '  
Escribir nombrePersona
```

Ejemplo en programa:

```
*** Ejecución Iniciada. ***  
¿Como te llamas?  
>   
■
```

```
*** Ejecución Iniciada. ***  
¿Como te llamas?  
> Alguien  
Hola Alguien  
*** Ejecución Finalizada. ***
```

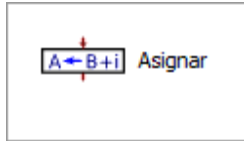
Como se puede ver, la variable no utiliza comillas, aquí un ejemplo de que ocurriría si utilizará comillas:

```
Escribir '¿Como te llamas?'  
Leer nombrePersona  
Escribir Sin Saltar 'Hola '  
Escribir 'nombrePersona'
```

```
*** Ejecución Iniciada. ***  
¿Como te llamas?  
> Alguien  
Hola nombrePersona  
*** Ejecución Finalizada. ***
```

Como se detalla en la imagen del programa, aparece como si 'nombre Persona' fuera un texto y no una variable con valor.

PSeInt: Comando Asignar



Asignar: Permite crear variables con valores predeterminados.

Desarrollo:

El nombre de la variable no puede utilizar comillas de ningún tipo, ejemplo:

Correcto:

- nombreVariable

Incorrecto:

- 'nombreVariable'
- "nombreVariable"

Para asignar un valor a una variable, de estar escrito primero el nombre de la variable, seguido de `<-` y por ultimo su valor, ejemplo:

(1) nombreVariable<-valor

(2) nombreVariable <- valor

(En (1) la `<-` está pegada y en (2) está separada)

El valor de la variable en texto debe ser utilizado con comillas, el numérico se recomienda que sea sin comillas, también puedes dar valor de una variable previa a una nueva y también que dependa de operadores/funciones matemáticas, ejemplo:

- variableTexto <- 'Texto'
- variableNumerica <- 1
- variableSuma <- variableNumerica + 2

Los nombres de las variables no pueden ser solo números, para poder utilizar números, previamente debe existir letra, es decir:

Correcto:	Incorrecto:
<ul style="list-style-type: none">• nombreVariable1 <- 1• nombre1Variable <- 1	<ul style="list-style-type: none">• 1 <- 1• 1nombreVariable <- 1

Para poder insertar una variable en un texto puede ser de la manera más cómoda, puede ser mediante una línea extra o con el uso de ',' (comas), ejemplo:

- variableNombre <- '¿cómo'
- Escribir 'Hola ', variable Nombre, ' estás?'
- variableNombre <- 'Hola'
- Escribir variableNombre, ' ¿cómo estás?'
- variableNombre <- '¿cómo estás?'
- Escribir 'Hola ', variable Nombre

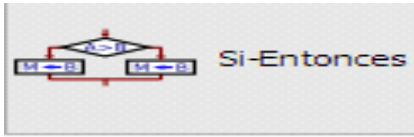
Ejemplo en programa:

```
saludo <- 'Hola'
Escribir saludo
numeroSuma <- 2
Escribir 'Indica un numero para sumarle +', numeroSuma
Leer numeroParaSumar
numeroTotal <- numeroParaSumar + numeroSuma
Escribir Sin Saltar 'El resultado es: '
Escribir numeroTotal
```

```
*** Ejecución Iniciada. ***
Hola
Indica un numero para sumarle +2
>
```

```
*** Ejecución Iniciada. ***
Hola
Indica un numero para sumarle +2
> 5
El resultado es: 7
*** Ejecución Finalizada. ***
```

PSeInt: Comando Si-Entonces



Si-Entonces: Este comando trabaja como condiciones, es decir, ‘Si’ condición previa se cumple ‘Entonces’ se hará una acción, esto trabaja como condición de valor de verdad (verdadero ‘v’ o falso ‘f’), si la condición se cumple (o sea es ‘v’) entonces se hace la acción en caso contrario es ‘f’

Desarrollo

Se debe escribir la palabra ‘Si’ seguida de la condición tras esta la palabra ‘Entonces’, ejemplo:
Si condición Entonces

Cuando la condición se cumple hará las acciones asignadas, al finalizar todas las acciones que se deberán hacer en este ‘Si-Entonces’, se debe terminar con un ‘FinSi’, ejemplo:

- Si condición Entonces
- Acciones

FinSi

En caso de querer dar una acción si la condición no se cumple, antes del ‘FinSi’ debe existir un ‘Sino’ seguido de sus respectivas acciones, ejemplo:

- Si condición Entonces
- Acciones si la condición se cumple

Sino

- Acciones si la condición no se cumple

Ejemplo en programa:

```
*** Ejecución Iniciada. ***
Hola
¿te llamas Jose? (si o no)
>
```

```
*** Ejecución Iniciada. ***
Hola
¿te llamas Jose? (si o no)
> si
Que bien! He adivinado
*** Ejecución Finalizada. ***
```

```
*** Ejecución Iniciada. ***
Hola
¿te llamas Jose? (si o no)
> no
Ah! Que mal, he fallado
*** Ejecución Finalizada. ***
```

```
Escribir 'Hola'
nombreAdivinar <- 'Jose'
Escribir '¿te llamas ',nombreAdivinar,'? (si o no)'
Leer respuestaUsuario
Si respuestaUsuario = 'si' Entonces
..... Escribir 'Que bien! He adivinado'
Sino
..... Escribir 'Ah! Que mal, he fallado'
FinSi
```

PSeInt: Comando Según



Según: Permite trabajar con un sistema de valor número, donde, según el valor elegido ocurría una secuencia de acciones.

Desarrollo:

- Debe existir previamente la variable con la que se trabajará, ya sea por el comando 'Leer', o por 'Asignar'
- La variable solo debe ser de valor numérico, no se permite del tipo texto

- Para comenzar el comando, este inicia con la palabra 'Segun' seguido de la variable numérica previamente existente y seguido de la palabra 'Hacer', ejemplo: Segun variable, Numerica Hacer
- Tras tener el comienzo del comando, prosigue una secuencia de acciones que trabajan con opciones/condiciones numéricas que las llaman.

Ejemplo en programa:

```

Escribir 'Seleccione:'
Escribir '(1) Saludo'
Escribir '(2,3) Nada'
Escribir '(4) Prueba'
Escribir '(5) Cerrar'
Escribir Sin Saltar 'Escriba el numero:'
Leer respuestaUsuario
Segun respuestaUsuario Hacer
    1:
        Escribir 'Hola, ¿como estas?'
        Leer saludoUsuario
        Escribir 'Presiona cualquier tecla para cerrar'
    2 , 3:
        Escribir 'No puedo nadar'
        Escribir 'Presiona cualquier tecla para cerrar'
    4:
        Escribir 'Esto es una prueba'
        Escribir 'Presiona cualquier tecla para cerrar'
De Otro Modo:
    Escribir 'Presiona cualquier tecla para cerrar'
Fin Segun

```

<pre> *** Ejecución Iniciada. *** Seleccione: (1) Saludo (2,3) Nada (4) Prueba (5) Cerrar Escriba el numero:> _ </pre>	<pre> *** Ejecución Iniciada. *** Seleccione: (1) Saludo (2,3) Nada (4) Prueba (5) Cerrar Escriba el numero:> 1 Hola, ¿como estas? > _ </pre>
---	---

<pre> *** Ejecución Iniciada. *** Seleccione: (1) Saludo (2,3) Nada (4) Prueba (5) Cerrar Escriba el numero:> 2 No puedo nadar Presiona cualquier tecla para cerrar *** Ejecución Finalizada. *** </pre>	<pre> *** Ejecución Iniciada. *** Seleccione: (1) Saludo (2,3) Nada (4) Prueba (5) Cerrar Escriba el numero:> 1 Hola, ¿como estas? > bien Presiona cualquier tecla para cerrar *** Ejecución Finalizada. *** </pre>
---	---

<pre>*** Ejecución Iniciada. *** Seleccione: (1) Saludo (2,3) Nada (4) Prueba (5) Cerrar Escriba el numero:> 3 No puedo nadar Presiona cualquier tecla para cerrar *** Ejecución Finalizada. ***</pre>	<pre>*** Ejecución Iniciada. *** Seleccione: (1) Saludo (2,3) Nada (4) Prueba (5) Cerrar Escriba el numero:> 4 Esto es una prueba Presiona cualquier tecla para cerrar *** Ejecución Finalizada. ***</pre>
---	---

```
*** Ejecución Iniciada. ***
Seleccione:
(1) Saludo
(2,3) Nada
(4) Prueba
(5) Cerrar
Escriba el numero:> 5
Presiona cualquier tecla para cerrar
*** Ejecución Finalizada. ***
```

PSeInt: Comando Mientras



Mientras: Permite crear un ciclo, que terminará cuando la condición no se cumpla

Desarrollo:

- Debe existir previamente una variable que será utilizada en la condición
- Para utilizar el comando, primero se debe escribir 'Mientras' seguido de la condición y luego la palabra 'Hacer', ejemplo: Mientras condición Hacer
- Luego del 'Hacer', debe seguir la secuencia de acciones mientras la condición se cumpla, y al final de estas cerrar el comando con 'Fin Mientras', ejemplo:
Mientras condición Y Hacer acciones.

Fin Mientras

Ninguna acción después del 'Fin Mientras' será ejecutada por el programa, hasta que el Mientras no se termine, es decir:

- Mientras condición Hacer

- acciones

Fin Mientras

Acciones que esperaran que termine el mientras

Ejemplo en programa:

```
variableCondicion <- 0
Mientras variableCondicion != 18 Hacer
  Escribir 'Cuanto es:'
  Escribir '2 + 4 x 5 - 12 / 3'
  Leer variableCondicion
  Si variableCondicion != 18 Entonces
    Escribir 'Te has equivocado, vuelve a intentarlo'
  FinSi
Fin Mientras
Escribir 'Felicidades has acertado!'
```

```
*** Ejecución Iniciada. ***
Cuanto es:
2 + 4 x 5 - 12 / 3
> 6
Te has equivocado, vuelve a intentarlo
Cuanto es:
2 + 4 x 5 - 12 / 3
> _
```

```
*** Ejecución Iniciada. ***
Cuanto es:
2 + 4 x 5 - 12 / 3
> 6
Te has equivocado, vuelve a intentarlo
Cuanto es:
2 + 4 x 5 - 12 / 3
> 18
Felicidades has acertado!
*** Ejecución Finalizada. ***
```

```
*** Ejecución Iniciada. ***
Cuanto es:
2 + 4 x 5 - 12 / 3
> _
```

PSeInt: Comando Repetir



Repetir: Permite hacer una secuencia de acciones hasta completar una condición

Desarrollo:

- Debe existir previamente una variable con el valor de inicio
- Debe existir un valor de llegada, ya sea mediante variable o predeterminado
- Para empezar el comando, se necesita escribir 'Repetir' luego colocar la secuencia de acciones y para finalizar colocar 'Hasta Que' y la condición para finalizar la repetición (Riaz, M., & Shehzad, S., 2018).

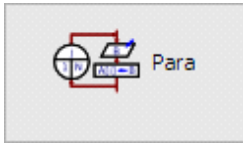
Ejemplo:

- Repetir
- acciones
- Hasta Que condición

Ejemplo en programa:

```
Escribir 'Coloca tu edad y por cada 4 años que tengas te daré un aplauso'
Leer respuestaUsuario
variableRepetir <- 0
variableLlegar <- trunc(respuestaUsuario/4)
Si variableLlegar >= 1 Entonces
  Repetir
    Escribir '*Aplauso*'
    variableRepetir <- variableRepetir + 1
  Hasta Que variableRepetir >= variableLlegar
FinSi
```

PSeInt: Comando Para



Para: Trabaja de manera similar al ‘Repetir’, solo que, aquí asignamos las variables dentro del comando.

Desarrollo:

- Debe existir previamente un valor de llegada
- Para comenzar el comando se debe escribir ‘Para’ seguido de un variable a la cual ahí mismo se le asigna un valor inicial, ejemplo:

Para variableInicio <- 1

- Luego se debe escribir ‘Hasta’ y el valor de llegada, este valor debe ser colocado predeterminado o proveniente de una variable anterior, ejemplo:

- Para variableInicio <- 1 Hasta 10

- variableLlegada = 10 Para variableInicio <- 1 Hasta variableLlegada

- Seguidamente hay que escribir ‘Con Paso’ e indicar el número de pasos que dará por cada repetición es decir, cuanto se le sumará al valor, y terminar con un ‘Hacer’ ejemplo:

Para variableInicio <- 1 Hasta 10 Con Paso 1 Hacer

- Después de ello, se debe colocar la secuencia de acciones y al final terminar el comando con un ‘Fin Para’, ejemplo:

Para variableInicio <- 1 Hasta 10 Con Paso 1 Hacer acciones

Fin Para

Ejemplo en programa:

<pre>*** Ejecución iniciada. *** Indique su edad Te daré aplausos por cada 4 años que tenga tu edad > 18 *Aplauso* *Aplauso* *Aplauso* *Aplauso* *** Ejecución finalizada. ***</pre>	<pre>*** Ejecución iniciada. *** Indique su edad Te daré aplausos por cada 4 años que tenga tu edad ></pre>
---	--

```

Escribir 'Indique su edad'
Escribir 'Te daré aplausos por cada 4 años que tenga tu edad'
Leer respuestaUsuario
variableLlegada <- trunc(respuestaUsuario / 4)
Para variableRepetir<-1 Hasta variableLlegada Con Paso 1 Hacer
..... Escribir '*Aplauso*'
Fin Para
```

2.2.7. Flujogramas y ejercicios

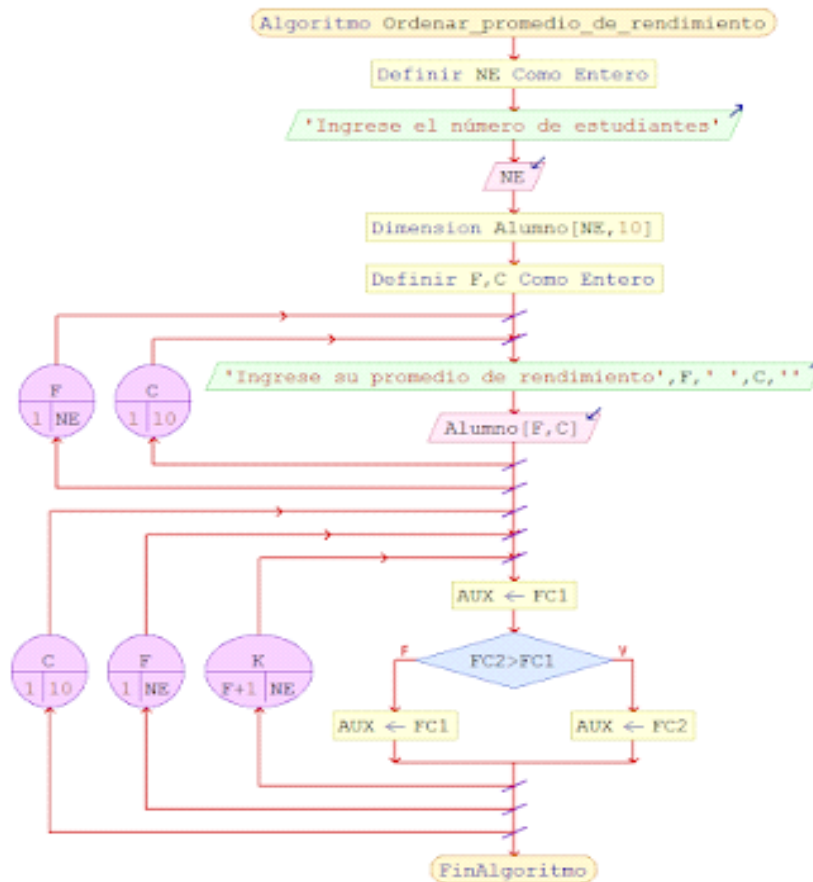


Figura 2.- Diagrama de Flujo en PSeInt
Elaborado por:- Autores de la Investigación

¿Por qué usar PSeInt y no otro intérprete o compilador de pseudocódigo?

- (1) Porque es software libre, sin necesidad de andar gastando dinero, haciendo giros, etc., violando los derechos de autor ni andar creando o consiguiendo cracs, que a veces sus links están inactivos y/o los programas no dejan craquearse.
- (2) Está constantemente atendido por su creador, a diferencia de los otros compiladores o intérpretes de pseudocódigo que están descontinuados.
- (3) Posee un foro para reportar errores y obtener ayuda, que está también constantemente atendido por su creador, esto ayuda a que los usuarios colaboren para corregir y mejorar el programa y colaboren entre ellos.

- (4) Posee una extensa ayuda, que valga la redundancia ayuda a aprender a usarlo, y a aprender el lenguaje.
- (5) Está disponible su código fuente, y con instrucciones para ejecutarlo, de modo que si sabemos C++ podremos modificarlo, para personalizarlo y/o corregirlo.
- (6) Posee previsualización y exportación a C, C++, Java y otros lenguajes para que podamos ver el mismo código implementado en un lenguaje de programación real, lo que ayuda a aprender estos y otros lenguajes.
- (7) Se trata de un compilador que compila automáticamente cuando el usuario pulsa ejecutar, el algoritmo se guarda automáticamente en un archivo del disco duro, para su posterior ejecución, haciendo más cómodo su uso.
- (8) Es un software multiplataforma, está disponible para Windows (a partir de Windows XP), GNU/Linux y Mac, de modo que podemos seguir utilizándolo pese a que eventualmente tengamos que utilizar otro sistema operativo que no sea Windows. También hay un desarrollo independiente para Android, que usa la sintaxis estricta de PSeInt.

A comparative study of teaching computational thinking with Scratch and PSeInt

(Kim, Y. J., & Kim, Y. J., 2018).

2.2.8. Funciones / Subprocesos en PSeInt

Las funciones también llamadas Subproceso o SubAlgoritmo, es una instrucción que permite agrupar variables y sentencias, cuya finalidad es la de ejecutar una tarea específica, se pueden añadir una o varias funciones, en un algoritmo, las funciones son subprogramas dentro de un programa, que se pueden invocar (ejecutar) desde cualquier parte del programa, es decir, desde otra función, desde la misma función o desde el programa principal, cuantas veces sea necesario (García-Peñalvo et al., 2019).

2.2 RAPTOR

RAPTOR es un entorno de programación icónico, diseñado específicamente para ayudar a los estudiantes a visualizar clases y métodos y limitar la complejidad sintáctica. Los programas RAPTOR se crean visualmente usando una combinación de UML y diagramas de flujo.

La herramienta RAPTOR es una zona de programación que permite llevar a cabo el desarrollo visual, haciendo uso de diagramas de flujo. Para ello se destaca que un diagrama de flujo se representa como un conjunto de símbolos gráficos los cuales presentan conexión, cada uno de estos símbolos presenta un objetivo específico por cumplir, llevan a cabo diversas instrucciones, es necesario que estos se presentan conectados para establecer un orden de cumplimiento de cada una de ellas, lo que permite que su funcionamiento se lleve a cabo según las características de Windows (Al-Aqrabi, H., & Almohri, 2018).

RAPTOR: es un programa que ayuda a los estudiantes a aprender a programar utilizando diagramas de flujo. Está diseñado para ser utilizado por principiantes y se enfoca en enseñar a los estudiantes cómo se relacionan los algoritmos y los diagramas de flujo.

RAPTOR: A visual programming environment for teaching algorithmic thinking
(Al-Aqrabi, H., & Almohri, 2018).

2.2.1. Funcionamiento del programa RAPTOR

El programa RAPTOR se presenta en funcionamiento en el sistema operativo Windows, este se encarga de llevar a cabo la creación de pseudocódigo a partir del uso de diagramas de flujo, a partir de ello se pueden llevar a cabo distintos procesos, verificaciones, realizar operaciones, y otros.

El software RAPTOR al ser iniciado se presenta con la apertura de dos pantallas, una de las pantallas presenta mayor tamaño y en ella es donde se presentan los procesos que se van a ir llevando a cabo, en la pantalla más pequeña se destaca el proceso ya realizado, este al ir apareciendo se presentará finalizado (Almohri, H., & Al-Aqrabi, H., 2019).

Al momento en que se lleva a cabo un diagrama de flujo es importante que siempre se presente un Inicio y un Fin, para que se puedan observar el momento en que se da comienzo el proceso y así

como cuando este termina, por lo tanto, se indica en el ‘*Start*’ y el ‘*End*’, estas son opciones que siempre aparecen en la ventana.

En el caso de los demás comando, estos no se muestran siempre en pantalla, estos son agregados en los procesos realizados, cuando estos se requieren para la solución de algún problema o la necesidad llevar a cabo alguna acción, de esta manera se le debe de dar doble clic en alguno de ellos para que pueda aparecer de color rojo indicando la acción de los mismos y el poder indicar en que parte del diagrama se llevará a cabo, este proceso se mantiene constante hasta el final, lo cual viene parte en las ventajas y desventajas de Windows.

2.2.2. ¿Por qué utilizar RAPTOR?

Las razones para utilizar RAPTOR son muy variadas. Si eres un programador primerizo y necesitas de una manera que te permita practicar tu lógica sin duda es una gran opción. Por otro lado, si eres un programador más avanzado y necesitas una forma rápida y sencilla de probar tus algoritmos también puede resultarte de utilidad (León, González y González, 2018, p. 1230).

Se trata de una herramienta de resolución de problemas que es muy fácil de utilizar, y es por esta razón que permite desarrollar distintas habilidades como la resolución de problemas y el pensamiento algorítmico (Cornelio et al., 2016). Además de esto si tienes experiencia anterior con diagramas de flujo podrás comenzar a utilizarlo de una forma muy rápida.

2.2.3. Estructura del programa RAPTOR

Siendo un conjunto de conexiones de símbolos, para que cada uno de ellos accionen según la tarea o instrucción que tienen, por lo tanto, se destaca que el programa RAPTOR posee flechas que indican estas conexiones y son los que destacan el orden que se va a llevar a cabo, en el momento en que se lleva a cabo la ejecución del programa RAPTOR es necesario acceder desde el inicio y seguir tal orden que indican las flechas para que de esta manera se pueda ejecutar el programa.

Programar

Cuando se lleva a cabo la descarga del programa RAPTOR se debe de instalar el mismo, como por ejemplo en un programa de Windows, luego de haber instalado se procede a abrir y dar apertura a una ventana. Se destacan tres secciones en el programa RAPTOR, los cuales son destacados.

Espacio de trabajo

Es la zona donde se lleva a cabo el proceso de trabajo, en él se presenta la creación de los diagramas a partir del uso de los símbolos que le componen.

Símbolos

Son las opciones que permiten que se lleven a cabo la creación de los diagramas, se presentan distintos tipos de símbolos donde cada uno de ellos cumple una función, los cuales serán detallados más adelante.

Barra de Menús

En esta barra se presenta una diversa cantidad de opciones que se pueden emplear en el proceso, entre ellas se encuentran:

- **Botones de Acceso Rápido:** Se presenta como parte de barra de herramientas donde ofrecen opciones con funciones relevantes que pueden ser usadas en ciertos momentos, entre las opciones se puede presentar: Abrir, Copiar, Guardar, Crear nuevo archivo, Detención, y otras.
- **Menú archivo:** Es el menú específico donde se presenta las opciones de creaciones de archivo, de abrir, guardar, acción de imprimir, compilar, así como también la posibilidad de poder guardar un diagrama directamente en una porta papel, lo cual es empleado para poder pegar este diagrama a un documento y poder guardarlo como una imagen.
- **Menú Editar:** Es un menú característico por ser de mucha utilidad, en él se pueden llevar a cabo la selección de opción que permite copiar y pegar ciertos fragmentos que se presentan en el diagrama y además también permite descartar cambios que se han llevado a cabo en el diagrama.
- **Menú Escala:** El uso de los menús de escala permiten llevar a cabo la acción de variación de tamaño del diagrama.
- **Menú Ver:** Haciendo uso de las opciones de este menú se podrán modificar la visualización en los diagramas, así como comentarios, variables.
- **Menú Correr:** El uso de este menú permite que se lleve a cabo la ejecución de la aplicación, este puede llevarse a cabo paso a paso, así como de un solo proceso.

- **Menú Modo:** En él se puede hacer la selección de los modos en que se desea llevar a cabo el proceso de funcionamiento, entre las opciones que se presentan son el de: ‘Principiante’, ‘Intermedio’ u otros.
- **Menú Tinta:** A partir del uso de este menú se puede seleccionar un color para la escritura a realizar, así como las anotaciones que se realizan y los subrayados que son necesarios, este color se presenta en todo punto del trabajo.
- **Menú Ventana:** Se encarga de llevar a cabo la orientación que se desea presentar ventana de la aplicación ejecutada esta puede ser tanto horizontal, como vertical o de forma completa.
- **Menú Generar:** Se presenta como entre las mejores opciones que destaca el programa RAPTOR, las cuales se pueden realizar por el programa creado, esto puede ser convertido en un lenguaje de programación, luego de que se escoge el lenguaje, a partir de ello se lleva a cabo la creación de un archivo que podrá ser editado en cualquier momento a partir de los diagramas usados.
- **Menú Ayuda:** En la opción de ‘Ayuda’ se presenta u manual de indicaciones para conocer la manera correcta de llevar a cabo operaciones, funciones, sintaxis (Muñoz-Pérez et al., 2019).

2.2.4. Características del programa RAPTOR

Entre las principales características del software RAPTOR se encuentran las siguientes:

Características de RAPTOR

- ❖ **Sin costo:** Se distribuye gratuitamente con el fin de apoyar a la comunidad de programadores que desean mejorar sus conocimientos.
- ❖ **Simbología común:** RAPTOR utiliza la misma simbología que los diagramas de flujo, lo que lo hace muy accesible para los estudiantes que deben tener estos conocimientos previamente.
- ❖ **Conversión a código:** Gracias a la herramienta para convertir los diagramas creados a código permite probar el resultado de los algoritmos diseñados y así comprobar su funcionamiento.

- ❖ **Es un entorno de programación:** Aunque no utiliza código como tal, su funcionamiento es muy similar a los entornos de programación por lo que prepara a los estudiantes de forma que puedan migrar a lenguajes más complejos de una forma sencilla.
- ❖ **Compatible con Windows:** Debido a que fue desarrollado utilizando el lenguaje C# su compatibilidad es exclusiva con Windows, y los desarrolladores no tienen planes para llevarlo a otros sistemas operativos por el momento.

RAPTOR as an educational tool for enhancing problem solving skills in students
(Kumar, A. S., & Parthiban, P.,2020).

2.2.5. Interfaz de trabajo

La interfaz gráfica del software permite la adición de comentarios a los símbolos del diagrama de flujo, por tanto, es posible editar el enunciado inicial del problema como un comentario del símbolo inicio (*Start*).

Los comentarios pueden estar en cualquier idioma, sin embargo, la herramienta está diseñada completamente para el idioma inglés. Desde nuestro punto de vista la herramienta RAPTOR tiene los siguientes inconvenientes:

- (1) Carece de soporte para el análisis del problema y planteamiento de la solución, aun cuando permite editar comentarios relacionados con los símbolos del diagrama.
- (2) No se presenta la traza completa de la ejecución en una pantalla que incluya la evaluación de las expresiones lógicas inmersas en las condiciones o ciclos, a pesar de que la prueba del algoritmo señala cuál es el símbolo en ejecución y se visualiza el cambio de valor de las variables.
- (3) No utiliza la notación estándar de la mayoría de los libros de algoritmos para representar la estructura cíclica *hacer hasta*.
- (4) No se realizan validaciones semánticas para evitar ciclos infinitos.
- (5) El símbolo de asignación solo acepta una asignación lo cual incrementa el tamaño de un diagrama.

- (6) No cuenta con las estructuras cíclicas para y mientras.
- (7) Solo se ejecuta de forma completa en plataformas Windows, para Linux debe instalarse la plataforma Mono.
- (8) El único idioma disponible es el inglés.

2.3. Modelos de programación

La elaboración de un algoritmo se puede realizar en forma totalmente libre, sin seguir los lineamientos de ningún modelo o en caso contrario aplicando las orientaciones de un prototipo (Marcillo, 2022; Sánchez & Barrezueta, 2022). En el desarrollo de un algoritmo siguiendo un modelo establecido podemos citar, entre otros, a los siguientes modelos:

2.3.1. Modelo declarativo

En este modelo se declara una serie de proposiciones, en general todo tipo de transformaciones que relatan el problema e individualizan su solución. Se indica qué es lo que se quiere obtener, no los pasos necesarios para obtener la solución. Se trabaja por medio de cláusulas que responden a la particularidad que afirmando se afirma, llamado *modus ponens* (latín). El modo *ponens* se basa en una regla de deducción que podría representarse de la siguiente manera: *Si proposicion1, entonces proposicion2*

Proposicion1 Por lo tanto, proposicion2

Por ejemplo, un razonamiento que sigue las inferencias del modo afirmando es:

“Si está nublado, entonces no llega el sol”

2.3.2. Modelo imperativo

En este modelo se detallan todos los pasos necesarios para encontrar la solución del problema. Las acciones se ejecutan secuencialmente, siguiendo una estructuración. Esta estructuración se implementa a través de las estructuras de control.

Ejemplo:

Inicio

- Acción 1
- Acción 2
- Acción 3

Hacer n veces

Si expresión es verdadera entonces

- Acción 4

- Acción 5
- Acción 6

Sino

- Acción 7
- Acción 8

Fin selección

Fin repetición

- Acción 9
- Acción 10

Fin

Nosotros utilizaremos este modelo imperativo aplicando una técnica específica que se desarrollará más adelante.

2.3.3. Modelo Orientado a Objetos

Los algoritmos que siguen este modelo se caracterizan porque tienen en cuenta las relaciones que existen entre todos los objetos que intervienen. Cada objeto o entidad que interviene en la solución tiene una determinada conducta, estado e identificación. En este modelo no se debe preguntar: ¿qué hace el algoritmo?, sino preguntar: ¿quién o qué lo hace? Algunos lenguajes de programación que siguen este modelo son: Smalltalk, C++, HTML, Java, etc.

2.4. Técnicas de programación

Muchas veces, encontrar la solución a un problema no resulta una tarea sencilla. Primeramente, hay que comprender bien cuál es el problema que se nos plantea, al cual le debemos encontrar su solución. En la actualidad, se debate constantemente sobre en qué lenguaje deberían concentrar sus esfuerzos aquellos estudiantes tratando de aprender a programar. Sin embargo, más que en el propio lenguaje, quizás la clave de aprender a programar sea aprender a pensar en clave de algoritmos.

Las técnicas de programación forman parte casi omnipresente de nuestras vidas: desde técnicas subconscientes mentales de programación hasta la programación informática. A continuación, echamos un vistazo a algunas técnicas de programación informática básicas.

2.4.1. Programación estructurada

La programación estructurada sigue tres reglas: la secuencia, la iteración y la decisión. Veamos cada una brevemente:

Secuencia

La secuencia en programación estructurada indica que las instrucciones del código se leerán de principio a fin desde la primera línea de código hasta la última, sin excepción.

Iteración

Indica que, según cierta condición, un número de instrucciones podrían repetirse un número determinado o incluso indeterminado de veces. Las iteraciones son básicamente estructuras cíclicas que nos permitirán repetir una cantidad de veces determinada o indeterminada unas instrucciones.

Decisión

La decisión o condición en programación estructurada indica que según unas ciertas condiciones dadas se ejecutarán o no un conjunto de instrucciones. Las condiciones permiten dividir nuestro código en "ramas", pudiendo así cambiar el flujo de ejecución, ejecutando algunas instrucciones o no según ciertas condiciones dadas (Renuka Devi, M. P., & Kamalakkannan, P., 2020).

- **Variables.** Las variables pueden considerarse las técnicas más fundamentales de programación. Su número y tipo depende del lenguaje que se esté utilizando.
- **Repeticiones o loops.** «*For*» es el tipo de repetición más extendido. Muchos lenguajes utilizan «*for*» para transmitir la idea de “*contar*”.
- **Decisiones o Selección.** Para que el programa sea flexible, debemos asegurarnos de que responda a la información introducida por el usuario. La mayoría de los lenguajes algorítmicos utilizan un método de selección para controlar el flujo del programa.
- **Arrays.** Los ‘*arrays*’ – o arreglos – son útiles para colecciones de elementos similares.
- **Aritmética modular.** Ayuda a limitar el número de salidas del programa o a hacer que las cosas “den la vuelta”. Es una de las técnicas de programación más sencillas y útiles.

- **Manipulación de texto.** El texto se almacena en forma de números. La capacidad de convertir símbolos en código ASCII y viceversa es una técnica extremadamente útil. Se puede aplicar en muchos casos. Uno de ellos es comprobar la presencia de mayúsculas y minúsculas, por ejemplo. Trocear líneas de código también es muy útil. De esta manera se pueden mostrar iniciales o crear anagramas.
- **Números aleatorios y escalados.** El escalado de números es muy útil, es una técnica que merece la pena dominar. La aleatoriedad también ayuda si queremos hacer que las cosas parezcan más naturales. Por ejemplo, aplicando técnicas de programación recursiva, has creado un árbol en *Scratch*, pero no tiene un aspecto natural. Añadiendo aleatoriedad se puede crear un efecto con aspecto más natural.

Esta técnica incorpora:

- **Diseño descendente (*top-down*):** el problema se descompone en etapas o estructuras jerárquicas.
- **Recursos abstractos (simplicidad):** consiste en descompones las acciones complejas en otras más simples capaces de ser resueltas con mayor facilidad.
- **Estructuras básicas:** existen tres tipos de estructuras básicas:
- **Estructuras secuenciales:** cada acción sigue a otra acción secuencialmente. La salida de una acción es la entrada de otra.
- **Estructuras selectivas:** en estas estructuras se evalúan las condiciones y en función del resultado de las mismas se realizan unas acciones u otras. Se utilizan expresiones lógicas.
- **Estructuras repetitivas:** son secuencias de instrucciones que se repiten un número determinado de veces.

Las principales ventajas de la programación estructurada son:

Ventajas de la programación estructurada

- ❖ Los programas son más fáciles de entender.
- ❖ Se reduce la complejidad de las pruebas.
- ❖ Aumenta la productividad del programador.
- ❖ Los programas queden mejor documentados internamente.

2.4.2. Programación modular

En la programación modular consta de varias secciones divididas de forma que interactúan a través de llamadas a procedimientos, que integran el programa en su totalidad. En la programación modular, el programa principal coordina las llamadas a los módulos secundarios y pasa los datos necesarios en forma de parámetros. A su vez cada módulo puede contener sus propios datos y llamar a otros módulos o funciones.

2.4.3. Programación Orientada a Objetos (POO)

Se trata de una técnica que aumenta considerablemente la velocidad de desarrollo de los programas gracias a la reutilización de los objetos.

El elemento principal de la programación orientada a objetos es el objeto. El objeto es un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Un objeto contiene varios datos bien estructurados y pueden ser visibles o no dependiendo del programador y las acciones del programa en ese momento.

El polimorfismo y la herencia son unas de sus principales características y por ello dedicaremos más adelante un artículo exclusivamente a tratar estos dos términos (Rodríguez et al., 2018).

2.4.4. Programación concurrente

Este tipo de programación se utiliza cuando tenemos que realizar varias acciones a la vez.

Se suele utilizar para controlar los accesos de usuarios y programas a un recurso de forma simultánea.

Se trata de una programación más lenta y laboriosa, obteniendo unos resultados lentos en las acciones.

2.4.5. Programación lógica

Se suele utilizar en la inteligencia artificial y pequeños programas infantiles. Se trata de una programación basada en el cálculo de predicados (una teoría matemática que permite lograr que

un ordenador basándose en hecho y reglas lógicas, pueda dar soluciones inteligentes) (Matos-Berrios, 2021).

2.5. ¿En qué profesiones hace falta saber programar?

El mundo cambia a toda velocidad. Cada día se publican opiniones sobre nuevos dispositivos y tecnologías. Hoy por hoy, existen muchas profesiones donde conviene tener conocimientos de desarrollo de código y programación – y no todas están relacionadas directamente con el mundo informático.

- **Programador.** Los programadores ganan buenos sueldos y la informatización universal incrementa la demanda de especialistas informáticos.
- **Desarrollador web.** La popularidad de las redes sociales, las ilimitadas posibilidades de Internet para los negocios, la educación, el entretenimiento – han convertido a la programación web en una profesión muy demandada, interesante y bien remunerada.
- **Copywriter, SMM – and optimizador SEO.** Estos perfiles profesionales están directamente relacionados con la informática – su trabajo es promocionar sitios en internet y las plataformas sociales. No desarrollan recursos web y bases de datos, pero estos profesionales tienen que entender cómo funcionan.
- **Contable, financiero.** Estos profesionales trabajan con muchos programas especializados, que con frecuencia son muy complejos y requieren ser configurados adecuadamente. Para ello necesitan disponer, al menos, de conocimientos básicos de desarrollo de código y programación (Guzmán et al., 2022).

De acuerdo con (Ling, et al., 2020), la capacidad de programar es extremadamente importante para cualquier persona, independientemente de la carrera profesional que hayan elegido. Aprendiendo a programar ejercitamos la memoria, nuestra capacidad intelectual y lógica y nos ayuda a alcanzar nuestras metas profesionales y vitales,

CAPÍTULO III. METODOLOGÍA

En este capítulo se describe la metodología utilizada en el proceso de investigación y formulación de resultados de este libro. Se detallan cada una de las técnicas y métodos de investigación.

3.1. Tipo de investigación

Investigación Exploratoria: La investigación explicativa fue aplicada para replicar las razones o motivos por los cuales es beneficiosa la utilización de las herramientas PSeInt y RAPTOR para la elaboración de algoritmos, observando las causas y los efectos que existen, como establecer una comparación entre el código que debe escribirse entre cada una de las herramientas.

3.2. Métodos

Para la ejecución del presente proyecto, fue necesaria la ayuda de métodos que permitan interpretar información relevante y fiable, para ello se hizo uso de los siguientes métodos:

3.2.1. Método Analítico-Sintético

El método analítico-sintético fue empleado mediante el análisis de la información actual sobre las herramientas PSeInt y RAPTOR, y su utilización para construir algoritmos con la ayuda de la sentencia repetitiva *While* en base a los algoritmos propuestos.

3.2.2. Método Bibliográfico

El método bibliográfico fue utilizado para sustentar el desarrollo del proyecto en base a los contenidos consultados y sistematizados, así como para delinear la utilización de la sentencia *While* en el desarrollo de los algoritmos propuestos.

3.3. Materiales

Para la realización de la investigación se hizo uso de los siguientes materiales o implementos:

- Computadora de escritorio/laptop.
- Teléfono celular.
- Herramienta PSeInt y RAPTOR
- Internet.

CAPÍTULO IV. RESULTADOS

4.1. Descripción de los resultados

En este capítulo se implementan cuatro ejercicios haciendo uso de las herramientas objeto de estudio: PSeInt y RAPTOR, para identificar las principales ventajas y desventajas de la utilización de cada una de estas.

4.2. Resolución de ejercicios

A. Programa que muestra los divisores exactos de un número

PSEINT

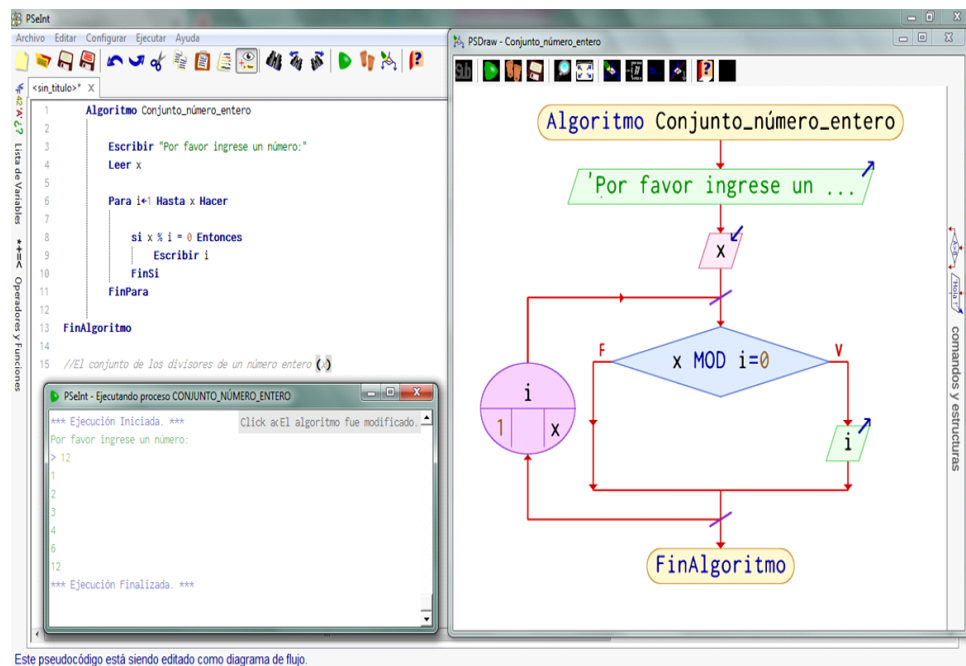


Figura 3.- Divisores exactos en PSeInt.

Elaborado por:- Autores de la Investigación.

RAPTOR

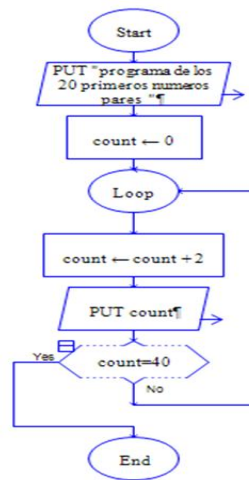


Figura 4.- Diseño en Raptor

Elaborado por: - Autores de la Investigación

- B.** Programa que genera una tabla de multiplicar de un número formado por la diferencia de dos números positivos. La tabla irá del 2 al 24 (de 2 en 2).

PSeInt

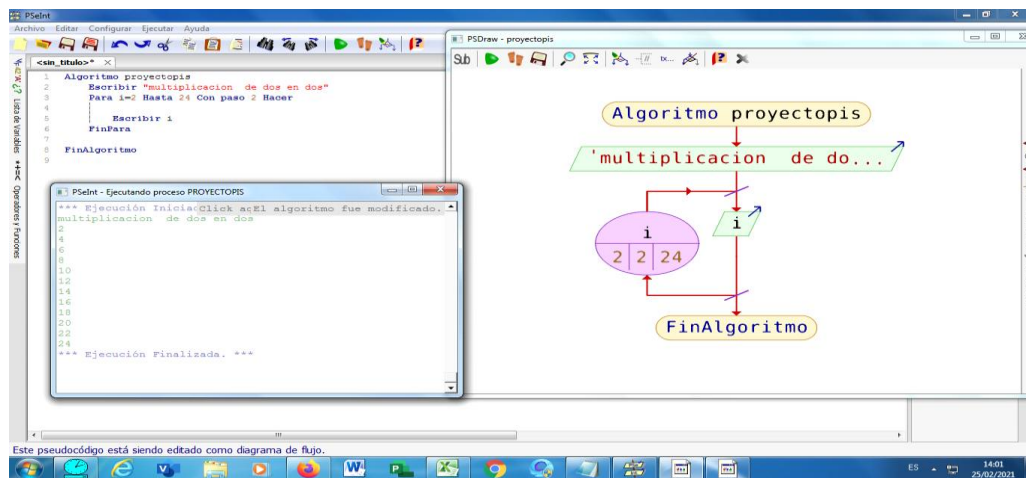


Figura 5.- Diferencias de números positivos

Elaborado por: - Autores de la Investigación

RAPTOR

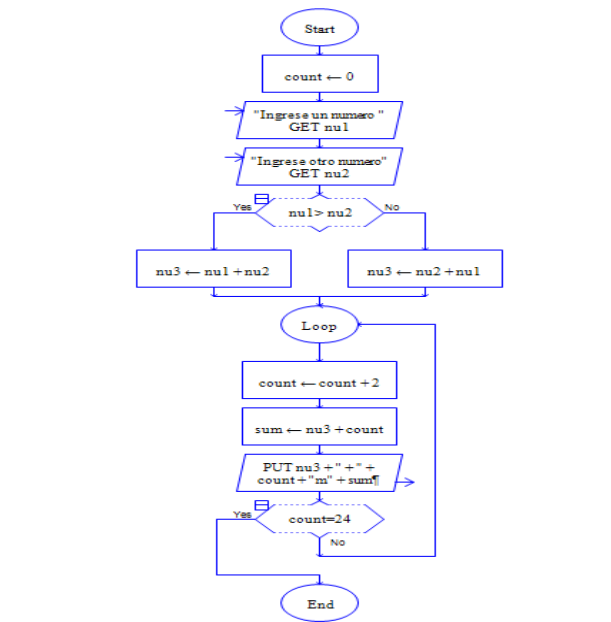


Figura 6.- Diagrama de flujo de números positivos.

Elaborado por: - Autores de la Investigación

C. Programa que muestra los N primeros números primos.

PSeInt

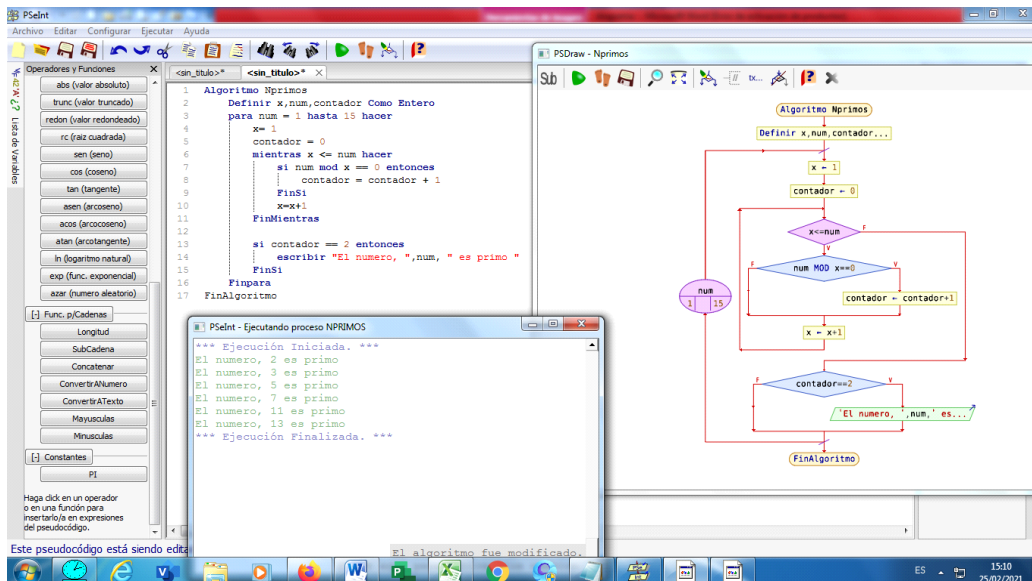


Figura 7.- Programa de N números primos.

Elaborado por: - Autores de la Investigación.

RAPTOR

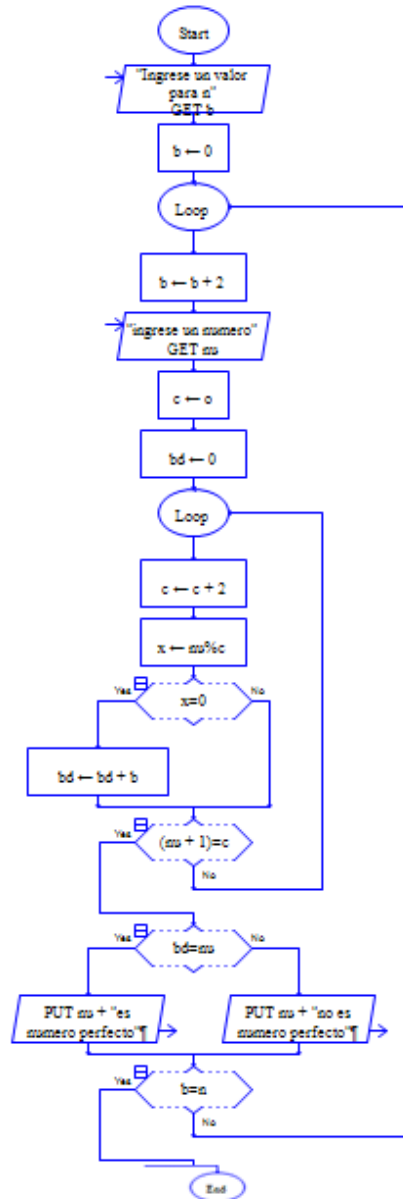
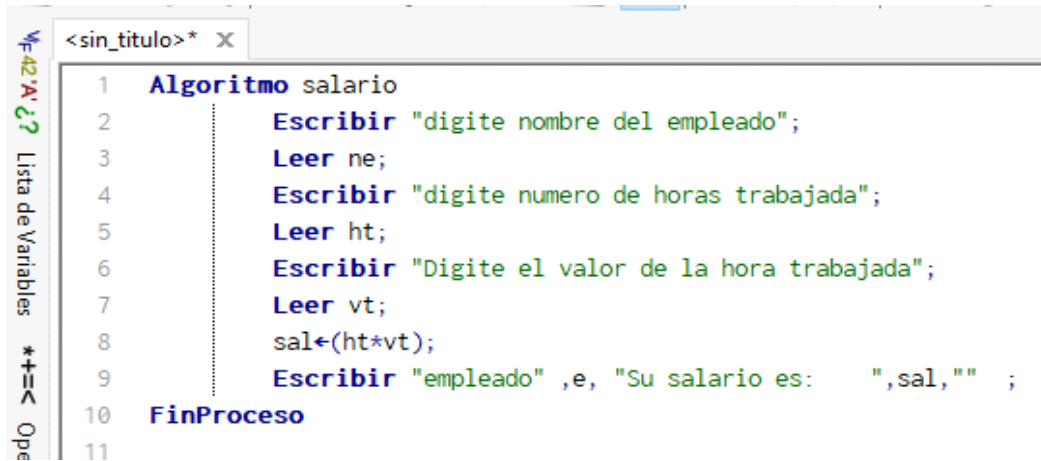


Figura 8.- Diagrama de Flujo de N números primos
Elaborado por: - Autores de la Investigación

- D. Programa que muestra el total de salarios de N empleados, de cada empleado se debe registrar el número de horas trabajadas y el valor por cada hora de trabajo.

PSeInt



```
<sin_titulo>* X
1  Algoritmo salario
2      Escribir "digite nombre del empleado";
3      Leer ne;
4      Escribir "digite numero de horas trabajada";
5      Leer ht;
6      Escribir "Digite el valor de la hora trabajada";
7      Leer vt;
8      sal←(ht*vt);
9      Escribir "empleado" ,e, "Su salario es:  ",sal,"" ;
10 FinProceso
11
```

Figura 9.- N salarios en PSeInt

Elaborado por: - Autores de la Investigación

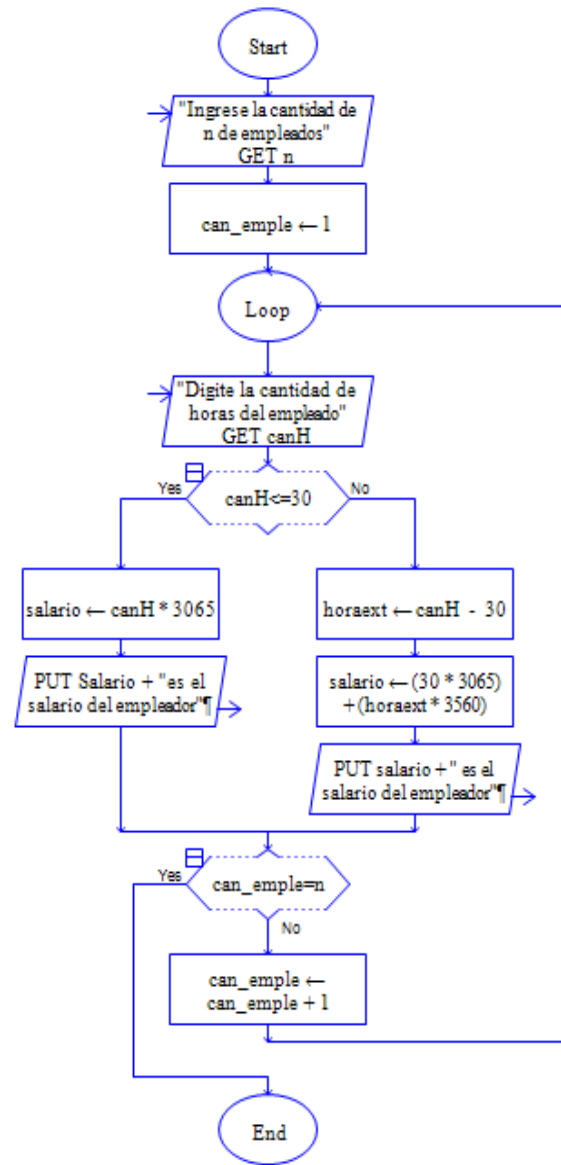


Figura 10.- Diagrama de Flujo de N salarios

Elaborado por: - Autores de la Investigación

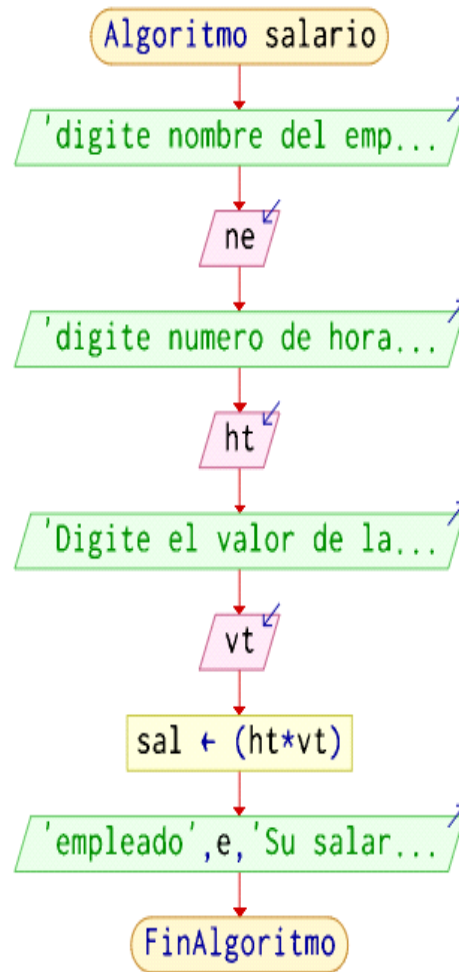


Figura 11.- Diagrama N salarios en PSeInt

Elaborado por: - Autores de la Investigación

4.3. Análisis de resultados

En base a los ejercicios resueltos en las dos herramientas, podemos establecer el siguiente análisis comparativo.

Tabla 1.- Análisis comparativo entre PSeInt y RAPTOR.

CARACTERÍSTICA	PSEINT	RAPTOR
Comprensión de código	Brinda facilidades para escribir y comprender el código, es muy parecido a lenguajes como el C.	No dispone de una interfaz para construir el código. El código se introduce al momento de utilizar las herramientas de diagramación.
Interpretación de diagrama	El diagrama es fácil de entender y se obtiene como parte del código. El diagrama representa la solución en símbolos dibujados con colores. El símbolo de Escritura y Entrada tienen el mismo aspecto, lo que podría causar confusiones en principiantes.	Al inicio es un poco compleja la utilización de los símbolos porque requieren de la incorporación de las variables y procesos que son parte de la solución.
Proceso de compilación	Dispone de un depurador de errores que indica la línea en la que se encuentra el error.	Este proceso es bastante interesante porque es visible la forma en la que la herramienta ejecuta cada paso del pseudocódigo, disponiendo de una ventana en la que se observan las variables y el

		valor que reciben en cada paso del pseudocódigo.
Ejecución del pseudocódigo	Es bastante sencilla, dispone de un icono en forma de “Play” con el cual se puede iniciar e interrumpir la ejecución. Los resultados se observan en una nueva ventana.	También es bastante sencilla, dispone de control de reproducción y pare de ejecución. A diferencia de PSeInt las entradas de datos ocurren al momento que el compilador interpreta el símbolo del diagrama.
Rendimiento/instalación	Cuenta con un buen rendimiento en todo momento. Su instalación es sencilla y amigable.	Cuenta con un buen rendimiento en todo momento. Su instalación es sencilla y amigable.
Aplicación de Estructura “Mientas”	Está muy bien definida y su representación y funcionalidad se da de acuerdo con la forma general.	No se dispone de un símbolo que de forma específica permita incorporar el ciclo mientras, solo dispone de un símbolo que permite realizar ciclos llamado LOOP.

Elaborado por: - Autores de la Investigación

Los resultados alcanzados en la investigación coinciden en gran medida con los resultados identificados en investigaciones similares:

En el trabajo realizado por (Albluwi & Aldossary, 2021, p. 36) El estudio mostró que RAPTOR resultó ser más efectivo para mejorar las habilidades de programación de los estudiantes que PSeInt. Sin embargo, (Chen, 2021, p. 23) reportó que los resultados del estudio indicaron que PSeInt fue más eficaz en la mejora de las habilidades de programación de los estudiantes.

Por otra parte, (Devi y Kamalakkannan, 2020) descubrieron que PSeInt era más efectivo para mejorar la comprensión de los conceptos de programación de los estudiantes (p. 401). Asimismo, (Al-Turaiki & Al-Ohali, 2019, p. 46) reporta que el estudio encontró que los estudiantes que usaron PSeInt se desempeñaron significativamente mejor en tareas de programación que aquellos que usaron el lenguaje C. Este resultado también fue similar al encontrado por Riaz y Shehzad (2018), en el cual el estudio reveló que PSeInt era más efectivo en la enseñanza de conceptos de programación a principiantes en comparación con el lenguaje de programación C.

Los resultados del estudio mostraron que tanto RAPTOR como PSeInt son herramientas efectivas para enseñar conceptos de programación, pero se encontró que PSeInt era más efectivo para mejorar las habilidades de programación de los estudiantes. Estos resultados también coinciden con los reportados por (Cruz-Vega & Guerrero-García, 2019, p. 3).

El estudio concluyó que tanto RAPTOR como PSeInt son efectivos en la enseñanza del pensamiento algorítmico, pero se encontró que PSeInt era más efectivo para mejorar las habilidades de resolución de problemas de los estudiantes.

CAPÍTULO V. CONCLUSIONES

La programación de computadoras es la principal demostración de las habilidades de pensamiento computacional. Sin embargo, tienden a seguir enfoques de enseñanza de programación orientados a la sintaxis, sin centrarse en el pensamiento computacional.

PSeInt es una herramienta adecuada para cursos de introducción a la programación por su sintaxis simple y facilidad para la depuración de código. La herramienta PSeInt permite que los estudiantes desarrollen habilidades de pensamiento algorítmico, lógica y estrategias de resolución de problemas mediante la creación de algoritmos con pseudocódigo. Además, por ser una herramienta que permite la ejecución de algoritmos de forma automatizada, los estudiantes pueden probar sus propuestas de solución, encontrar y resolver errores de lógica, practicando automatización y depuración, que son parte de las habilidades de pensamiento computacional.

RAPTOR es una herramienta de código abierto que admite completamente la programación orientada a objetos, incluida la encapsulación, la herencia y el polimorfismo. RAPTOR permite a los estudiantes ejecutar sus algoritmos dentro del entorno, en lugar de tener que compilar y ejecutar sus programas por separado. Esto significa que la depuración se puede realizar en la representación visual del algoritmo, en lugar de la textual, y evita tener que usar varias herramientas.

En base a los objetivos planteados se realizan las siguientes observaciones:

- PSeInt como RAPTOR son una herramienta muy útil para la representación de flujograma.
- PSeInt es la representación de pseudocódigo se utilizan palabras claves que se representan las acciones u operaciones. se caracteriza por mostrar donde hay errores en el pseudocódigo.
- RAPTOR es la representación de diagrama de flujo se utiliza simbología para el desarrollo del ejercicio.
- Desarrollamos ejercicios en las dos herramientas para apoyar en el análisis comparativo con la estructura WHILE (mientras).

RECOMENDACIONES

En base a los objetivos planteados en el presente trabajo de titulación se proponen las siguientes recomendaciones:

- Que los estudiantes conozcan el funcionamiento y utilización de la estructura WHILE en la herramienta PSeInt y RAPTOR.
- Se recomienda despertar el interés de los estudiantes a través de las herramientas estudiadas llevando más a la práctica de PSeInt y RAPTOR.

REFERENCIAS BIBLIOGRÁFICAS

- Abdallah, I. (2018). Teaching programming with PSeInt and Scratch: A comparative study. *International Journal of Emerging Technologies in Learning (iJET)*, 13(3), 142-153. <https://doi.org/10.3991/ijet.v13i03.8052>
- Agarwal, N., Bali, R. K., & Gupta, P. (2018). RAPTOR: A tool for teaching computational thinking. *Journal of Computing in Higher Education*, 30(1), 83-96. <https://doi.org/10.1007/s12528-017-9143-4>
- Al-Aqrabi, H., & Almohri, H. (2018). RAPTOR: A visual programming environment for teaching algorithmic thinking. *Education and Information Technologies*, 23(6), 2641-2659. <https://doi.org/10.1007/s10639-018-9754-4>
- Albluwi, I., & Aldossary, O. (2021). Evaluation of programming education tools: Comparative study of RAPTOR and Scratch. *International Journal of Emerging Technologies in Learning (iJET)*, 16(1), 29-39. <https://doi.org/10.3991/ijet.v16i01.11732>
- Almohri, H., & Al-Aqrabi, H. (2019). Teaching programming fundamentals using RAPTOR: A systematic literature review. *Education and Information Technologies*, 24(4), 2287-2314. <https://doi.org/10.1007/s10639-019-09883-7>
- Al-Turaiki, I., & Al-Ohali, Y. (2019). A comparative study of the effects of using PSeInt and C language in programming education. *International Journal of Emerging Technologies in Learning (iJET)*, 14(8), 41-49. <https://doi.org/10.3991/ijet.v14i08.10490>
- Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., & Yovine, S. (1995). The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1), 3-34. <https://www.sciencedirect.com/science/article/pii/030439759400202T>
- Álava, W. L. S., Rodríguez, A. R., Ávila, X. L. A., & Cornelio, O. M. (2022a). Impacto del uso de la tecnología en la formación integral de los estudiantes de la carrera tecnologías de la información. *Journal TechInnovation*, 1(2), 71-77. <https://revistas.unesum.edu.ec/JTI/index.php/JTI/article/view/21>

- Álava, W. L. S., Rodríguez, A. R., Ávila, X. L. A., & Cornelio, O. M. (2022b). Redes inalámbricas, su incidencia en la privacidad de la información. *Journal TechInnovation*, 1(2), 104-109. <https://revistas.unesum.edu.ec/JTI/index.php/JTI/article/view/25>
- Castillo-Barrera, F. E., Arjona-Villicana, P. D., Ramirez-Gamez, C. A., Hernandez-Castro, F. E., & Sadjadi, S. M. (2013). Turtles, Robots, Sheep, Cats, Languages what is next to teach programming? A future developer's crisis? *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*,
- Castrillón, A. M. S., Castrillon, A. S., Edison, T., & Barbosa, G. (2022). Analysis Of The Inclusion Of Pseint As An Initial Programming Language In South America. *Webology* (ISSN: 1735-188X), 19(6). [https://www.webology.org/data-cms/articles/20221109082315pmwebology%2019%20\(6\)%20-%2016.pdf](https://www.webology.org/data-cms/articles/20221109082315pmwebology%2019%20(6)%20-%2016.pdf)
- Cooper, G. (2014). Using Visual Logic with Pseudocode to Teach an Introductory Programming Course. *proceedings of WorldComp 2014*. <http://world-comp.org/preproc2014/FEC2198.pdf>
- Cornelio, O. M., Ching, I. S., Fonseca, B. B., & Díaz, P. M. P. (2016). Herramienta para la simulación de sistemas dinámicos integrado al sistema de laboratorios virtuales ya distancia. *Anais do Encontro Virtual de Documentação em Software Livre e Congresso Internacional de Linguagem e Tecnologia Online*
- Cruz-Vega, J., & Guerrero-García, J. (2019). A comparative analysis between two free tools for programming education: Scratch and PSeInt. In *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1-6). IEEE. <https://doi.org/10.23919/CISTI.2019.8760916>
- Chen, J. (2021). A comparative study of the effectiveness of PSeInt and Scratch in programming education. *International Journal of Emerging Technologies in Learning (iJET)*, 16(1), 19-28. <https://doi.org/10.3991/ijet.v16i01.11840>
- Del Prado, A., & Lamas, N. (2014). Alternativas para la enseñanza de pseudocódigo y diagrama de flujo. *Rev. Electrónica Iberoam. Educ. en Ciencias y Tecnol*, 5(3), 102-113. <https://exactas.unca.edu.ar/riecyt/VOL%205%20NUM%203/F%20%20SI%203%2014%20Trabajo%20Completo%20Fundamentos.pdf>

- Fernández-Sanz, L., & Escalona, M. J. (2021). A Comparative Analysis of Visual Programming Languages for Learning Programming Concepts. *IEEE Access*, 9, 35491-35503. <https://doi.org/10.1109/access.2021.3062039>
- García-Peñalvo, F. J., González Crespo, R., & Cruz-Benito, J. (2019). Comparing PSeInt with other tools for teaching programming fundamentals. *Journal of Educational Computing Research*, 57(3), 561-582. <https://doi.org/10.1177/0735633118760311>
- Guzmán, R. S. H., De La Rosa, C. G. B., Barrezueta, L. D. R., & Sánchez, P. M. M. (2022). Fundamentos de la auditoría: Una aproximación del estado del arte. *Serie Científica de la Universidad de las Ciencias Informáticas*, 15(12), 245-266. <https://publicaciones.uci.cu/index.php/serie/article/view/1282>
- Kim, Y. J., & Kim, Y. J. (2018). A comparative study of teaching computational thinking with Scratch and PSeInt. *Journal of Educational Technology & Society*,
- Kumar, A. S., & Parthiban, P. (2020). RAPTOR as an educational tool for enhancing problem solving skills in students. *Journal of Computing in Higher Education*, 32(1), 127-141. <https://doi.org/10.1007/s12528-019-09235-y>
- Laura-Ochoa, L., & Bedregal-Alpaca, N. (2022). Incorporation of computational thinking practices to enhance learning in a programming course. *International Journal of Advanced Computer Science and Applications*, 13(2). https://www.researchgate.net/profile/Leticia-Ochoa/publication/358926606_Incorporation_of_Computational_Thinking_Practices_to_Enhance_Learning_in_a_Programming_Course
- León, J. C., González, A. L., & González, R. E. (2018). Teaching algorithmic thinking with Scratch and PSeInt: A comparative study. In *2018 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1226-1232). IEEE. <https://doi.org/10.1109/EDUCON.2018.8363465>
- Ling, C. X., Zhao, H., & Yue, X. (2020). A comparative analysis of RAPTOR and Python for teaching algorithm design. *Journal of Computing in Higher Education*, 32(1), 1-18. <https://doi.org/10.1007/s12528-019-09220-5>
- Manso, A., Marques, C. G., Santos, P., Lopes, L., & Guedes, R. (2019). Algorithmi IDE-Integrated learning environment for the teaching and learning of algorithmics. *2019 International Symposium on Computers in Education (SIIE)*,

- Marcillo, P. M. (2022). Análisis del desarrollo de software con metodología ágil y la capacidad de la sostenibilidad implementada [Universidad Politécnica de Madrid]. <https://oa.upm.es/id/eprint/71758>
- Matos-Berrios, V. M., Martínez-García, A. M., Ortiz-Vergara, M. E., & Fernández-Álvarez, M. (2021). Programming education using PSeInt: An evaluation of student learning. *Journal of Educational Technology & Society*, 24(3), 130-142. <https://www.jstor.org/stable/26924905>
- Muñoz-Pérez, A. B., Ochoa-Ortega, C. A., & Ruíz-Vanoye, J. A. (2019). PSeInt vs Scratch: A comparative study for the teaching of programming fundamentals. *Journal of Educational Computing Research*, 56(7), 1041-1061. <https://doi.org/10.1177/0735633118778121>
- Pérez, M. A. G., Vega-Rodríguez, M. A., & Ayala-Ramírez, S. (2018). PSeInt as a learning tool for programming fundamentals. *International Journal of Engineering Education*
- Renuka Devi, M. P., & Kamalakkannan, P. (2020). A comparative study on the effectiveness of teaching programming using PSeInt and Scratch. *International Journal of Computer Applications Technology and Research*, 9(6), 400-405. <http://ijcat.com/archives/volume9/issue6/ijcatr09060801.pdf>
- Riaz, M., & Shehzad, S. (2018). Comparative analysis of PSeInt and C programming language for teaching basic programming concepts. *Bulletin of Electrical Engineering and Informatics*, 7(3), 403-411. <https://doi.org/10.11591/eei.v7i3.1154>
- Rodríguez, M. P., Salinas, D. D., & Acosta, D. A. (2018). An educational comparison of visual programming languages for introductory programming courses. *Journal of Information Systems Engineering & Management*, 3(4), 17. <https://doi.org/10.20897/jisem.201803>
- Sánchez, P. M. M., & Barrezueta, L. D. R. (2022). Análisis de la información generada para mantener la escalabilidad y persistencia del proceso de desarrollo de software. *Serie Científica de la Universidad de las Ciencias Informáticas*, 15(8), 193-227. <https://publicaciones.uci.cu/index.php/serie/article/view/1137>
- Ulate-Caballero, B. A., Berrocal-Rojas, A., & Hidalgo-Céspedes, J. (2021). Concurrent and Distributed Pseudocode: A Systematic Literature Review. 2021 XLVII Latin American Computing Conference (CLEI),

- Verner, I. M., & Card, D. N. (2018). The Power of Visualization in Teaching Computer Science. *Communications of the ACM*, 61(2), 49-54. <https://doi.org/10.1145/3173017>
- Viana, J. P. S., & Pilla, A. M. (2021). PSeInt as a tool for teaching algorithmic thinking. *IEEE Latin America Transactions*, 19(5), 731-737. <https://doi.org/10.1109/TLA.2021.9467148>

EDITORIAL

Cornelio, O. M., Ching, I. S., Fonseca, B. B., & Díaz, P. M. P. (2016). Herramienta para la simulación de sistemas dinámicos integrado al sistema de laboratorios virtuales ya distancia. Anais do Encontro Virtual de Documentação em Software Livre e Congresso Internacional de Linguagem e Tecnologia Online,

ISBN: 978-9942-7090-6-6



9 789942 709066