Transferencia de Conceptos Básicos de Programación de la Escuela Media a la Universidad

Transfer of Programming Basics from Middle School to College

Verónica Sofía D'Angelo [1, 2], Pedro Daniel López [1], Alejandro Mario Hernández [1, 3]

1 Centro de Altos Estudios de Tecnología Informática (CAETI), Universidad Abierta Interamericana, Rosario, Argentina.
2 Instituto Rosario de Investigaciones en Ciencias de la Educación (IRICE: CONICET-UNR), Rosario, Argentina.
3 Facultad de Ciencias Exactas, Ingeniería y Agrimensura (FCEIA: UNR), Rosario, Argentina.

dangelo@irice-conicet.gov.ar

Resumen. El estudio se centra en evaluar la transferencia de conceptos básicos de programación desde la escuela media a la universidad de estudiantes ingresantes a la carrera de Ingeniería en Sistemas de Información. Se realizan encuestas antes y después de un curso introductorio de programación. Los resultados parciales muestran una mejora significativa en la identificación de conceptos básicos después del curso, aunque inicialmente existe una baja comprensión de estos conceptos. Se destaca la relación entre el conocimiento verbal y procedural, con una brecha que disminuye tras el curso. Se evidencia una preferencia de los estudiantes por aprender programación con lenguajes basados en texto. Se concluye que es crucial integrar los conocimientos previos en la enseñanza universitaria y promover una comprensión realista de las carreras de informática desde la educación secundaria. Este estudio ofrece direcciones para futuras investigaciones en el campo del pensamiento computacional y la evaluación de conceptos adquiridos en programación.

Palabras claves: Algoritmos y Estructuras de Datos, Ciencias de la Computación, Psicología Cognitiva, Programación, Conceptos Básicos.

Abstract. The study focuses on evaluating the transfer of basic programming concepts from middle school to university. Two groups of students entering the Information Systems Engineering career at two different universities are analyzed. Surveys are conducted before and after an introductory programming course. The results show a significant improvement in the identification of basic concepts after the course, although initially there is a low understanding of these concepts. The relationship between verbal and procedural knowledge is highlighted, with a gap that decreases after the course. A preference among students for learning programming with text-based languages is evident. It is

concluded that it is crucial to integrate prior knowledge into university teaching and promote a realistic understanding of computer science careers from secondary education. This study offers directions for future research in the field of computational thinking and the evaluation of concepts acquired in programming.

Keywords: Introduction to Engineering, Programming, Robotics, Fab Lab, Spacemakers, Mechanical Engineering, Workshop modality, Arduino.

1 Introducción

Las ingenierías son fundamentales para el desarrollo industrial y tecnológico de un país. Con el continuo crecimiento y la creciente complejidad de los sistemas de software, se ha generado una mayor demanda no solo de ingenieros del software capacitados, sino también de un conocimiento más amplio y diversificado en áreas temáticas adicionales, como pruebas de software, inteligencia artificial, nuevos paradigmas, entre otros. Surge así la pregunta sobre cuándo y cómo introducir estos temas en el plan de estudios universitario.

Especialmente en el ámbito de la programación, el panorama ha evolucionado drásticamente en las últimas décadas. Lo que un estudiante de programación necesitaba dominar al principio de la carrera hace veinte años se limitaba principalmente a estructuras de control y algoritmos básicos. Sin embargo, en la actualidad, se espera que los estudiantes comiencen a adquirir un conocimiento más amplio, incluyendo algunas nociones de programación orientada a objetos, cierto manejo elemental de repositorios como GitHub y familiaridad con diversas plataformas de desarrollo. Resulta evidente que el primer año universitario no puede abordar todos estos temas de manera exhaustiva. Por lo tanto, es fundamental involucrar a la educación secundaria en la promoción de vocaciones y la preparación para carreras en ciencias de la computación (en adelante, CC).

1.1 Contexto internacional y local

Durante la segunda década del siglo XXI, se produjo un incremento significativo de las iniciativas destinadas a establecer las CC como parte integral del currículo escolar de secundaria a nivel mundial [1] –[8] y en Argentina [9].

En este contexto, se han marcado dos distinciones y "reclamos" de relevancia. En primer lugar, se enfatizó la distinción entre las CC y las Tecnologías de la Información y la Comunicación (TIC), subrayando que las CC poseen un carácter más específico, científico y menos utilitario en comparación con las TIC. Además, se insiste en que son las CC las que se procuran promover en la educación secundaria. En segundo lugar, se hace hincapié en la distinción entre las CC y la programación, reconociendo que esta última constituye sólo un componente básico aunque fundamental para las CC. Es importante tener en cuenta que este proceso de diferenciación no ha sido uniforme a

nivel internacional y, en el caso particular de Argentina, tampoco ha sido uniforme a nivel nacional.

Desde el año 2015, el Consejo Federal de Educación ha establecido la enseñanza de la programación como un área de interés estratégico en el sistema educativo. Según lo dispuesto en la Ley de Educación Nacional (Ley 26.206, en sus Disposiciones Generales de 2006), cada jurisdicción cuenta con la facultad de ajustar la estructura curricular de la enseñanza de la programación a su contexto específico, basándose en los lineamientos generales establecidos a nivel nacional. Esto implica que las instituciones educativas poseen cierta autonomía para determinar los contenidos, asignar prioridades y establecer obligatoriedades dentro del currículo computacional.

A pesar de que la Resolución 263/15 del Consejo Federal de Educación de Argentina aprobó en el año 2018 los "Núcleos de Aprendizajes Prioritarios de Educación Digital, Programación y Robótica" para todos los niveles educativos, incluyendo la escuela secundaria, y que dichos núcleos básicos contemplan el desarrollo de habilidades "analíticas, de resolución de problemas y de diseño para desarrollar proyectos de robótica o programación física, de modo autónomo, crítico y responsable…" [10, p.20], la naturaleza general de la resolución y la flexibilidad de adaptación han dado lugar a una falta de homogeneidad en su implementación.

Investigaciones recientes en nuestro país [11], [12] han examinado los diseños curriculares de escuelas secundarias no técnicas en 24 jurisdicciones, incluyendo 23 provincias y la Ciudad Autónoma de Buenos Aires, revelando resultados significativos. A pesar de las iniciativas y políticas públicas concretas de los últimos años, se ha observado un alto grado de disparidad en cuanto al lugar que ocupan las CC en el currículo, con una predominancia de los enfoques integradores, que buscan incorporar las TIC en otros espacios curriculares, pero una menor participación de enfoques específicos, que procuran posicionar la computación como una disciplina independiente, centrándose en sus propios contenidos y conceptos.

Se ha sugerido que esta disparidad puede atribuirse, en parte, a la falta de capacitación en informática por parte de los docentes, lo que limita su capacidad para proporcionar una formación más profunda en CC a sus estudiantes.

En la provincia de Santa Fe, si bien se realizaron experiencias que incluyen el desarrollo de una carrera de especialización docente en informática para nivel primario, la introducción de los contenidos en escuela media no cuenta aún con una especialización docente para dicho nivel. Existen, en la ciudad de Rosario, colegios pre univesitarios que incluyen desde hace décadas, contenidos de informática en sus planes, con independencia de las nuevas iniciativas. Sin embargo, el ciclo secundario, incluyendo el tramo orientado y los espacios de transición entre escuela media y universidad, continúa siendo un espacio desatendido. Paradójicamente, todo el esfuerzo por difundir el pensamiento computacional no está claramente articulado con la universidad en apoyo del estudiantado que precisamente, optó por seguir una carrera informática.

Lo que esto trae como consecuencia para todas las universidades tecnológicas es que en las carreras especializadas en informática se recibe un alumnado con gran disparidad de saberes, muchos de estos estudiantes presentan lagunas en contenidos básicos, a pesar de su motivación para aprender, evidenciada por la elección de la carrera. Esta disparidad conlleva a menudo a una sensación de frustración entre los estudiantes,

especialmente al final del primer año académico. Abordar este problema estructural de nivelación no es una tarea que pueda ser completamente resuelta durante los cursos de ingreso o al comienzo de las clases de programación. Requiere estrategias más amplias y sostenidas para su abordaje efectivo.

Es prudente considerar las lecciones aprendidas de experiencias pasadas en países que han enfrentado y superado dificultades similares. Por ejemplo, en Inglaterra se abogó por un mayor compromiso con las CC como disciplina independiente reconociendo la necesidad de cultivar habilidades específicas y conocimientos profundos en este campo [2]. Por otro lado, Australia ha realizado exploraciones profundas sobre los paradigmas pedagógicos del constructivismo y las ciencias cognitivas, con el objetivo de integrar y aprovechar las fortalezas de ambos enfoques para enriquecer la enseñanza en disciplinas STEM. Esta integración ha sido ejemplificada por la incorporación de la teoría de la carga cognitiva como un complemento de las teorías constructivistas, lo que demuestra un enfoque holístico y adaptativo hacia la educación en ciencias y tecnología [13], [14].

Siguiendo el modelo de los investigadores australianos, en esta investigación se buscó integrar algunos aspectos de la psicología cognitiva que se consideran útiles en la enseñanza del pensamiento computacional.

2 Objetivos

El objetivo de esta investigación, que se circunscribe al area de programación, fue evaluar el nivel de transferencia de conceptos básicos de programación en la transición desde la escuela media a la universidad. Para cumplirlo, es necesario analizar, en primer lugar, qué conceptos básicos de programación fueron efectivamente adquiridos en la escuela media, y, en segundo lugar, cuáles de estos conceptos, y de qué manera, son recordados y aplicados en las primeras experiencias de programación en la universidad. Siguiendo a Grover [18], proponemos evaluar el conocimiento de conceptos de programación a través de las descripciones verbales de los estudiantes.

Las dos preguntas de investigación generales se orientan a evaluar aspectos verbales y procedurales: 1. Evaluar qué conceptos los estudiantes ingresantes son capaces de reconocer por haberlos escuchado o leído durante el ciclo secundario y 2. Evaluar qué conceptos los estudiantes ingresantes son capaces de reconocer por haberlos escuchado o leído por primera vez en la universidad. Dicho reconocimiento se evaluará según lo que el estudiante es capaz de expresar verbalmente o reconocer en un fragmento de código.

3 Método

Aunque existen revisiones sistemáticas que abordan los distintos métodos de evaluación para medir el conocimiento de conceptos básicos de programación y del pensamiento computacional en general [15], es pertinente destacar que los estudios previos han tendido a enfocarse en la efectividad de intervenciones pedagógicas únicas en grupos específicos de estudiantes.

En contraste con estas investigaciones anteriores, el presente estudio se propone analizar las nociones básicas sobre programación que los estudiantes pueden haber adquirido en diversas escuelas secundarias, considerando la variabilidad en la dedicación a la programación de cada una. Para ello, se analizará un conjunto de estudiantes que ingresan a la carrera de ingeniería en sistemas de información en dos universidades distintas. Es importante tener en cuenta que estos estudiantes provienen de diferentes antecedentes educativos, con una heterogeneidad significativa en cuanto a su exposición y énfasis en la enseñanza de la programación durante la educación secundaria. El único punto en común que se presume entre estos contextos educativos es que los estudiantes, como mínimo, deberían tener cierto grado de familiaridad con nociones básicas del ámbito de la programación que les permitan expresar verbalmente términos técnicos (en alguna de sus diversas acepciones) y reconocerlos en fragmentos de código.
Dado que el objetivo de las preguntas de investigación es analizar el conocimiento conceptual que los estudiantes son capaces de expresar, antes y después de cursar 8 semanas de la asignatura de programación, se realizó una encuesta antes y otra después del cursado de dicha asignatura.

3.1 Participantes

Noventa y cuatro estudiantes de una asignatura introductoria a la programación, provenientes de dos universidades, una universidad privada (N=45) y una universidad pública (N=49) ambas de la ciudad de Rosario, Argentina, en la carrera Ingeniería en Sistemas de Información, (media de edad = 20.8 años; DE = 3.08), se inscribieron para cursar la asignatura Introducción a los algoritmos y la programación (Algoritmos y estructuras de datos, en la universidad pública).

3.2 Procedimiento

El contenido general de las clases era típico para una asignatura de programación estructurada introductoria, y coincidía en gran medida en ambas universidades. Se utilizaba *pseudocódigo* para resolver los algoritmos sin codificación en un lenguaje de programación, los algoritmos se resolvían mediante programación estructurada. Las clases constaban de dos horas de teoría y dos horas de práctica semanales. En las horas de teoría, se introducían conceptos y se ejemplificaban. Las horas de práctica los estudiantes resolvían ejercicios o presentaban resoluciones propias para contrastarlas con las soluciones que brindaba el profesor, con base en los conceptos dados en teoría.
El primer autor confeccionó la encuesta que fue distribuida por los profesores de la asignatura antes de iniciar el cursado (primer clase) y al final de la octava clase. Por cuestiones de infraestructura, las condiciones de la realización de la encuesta no fueron idénticas. En el caso de la universidad privada, el profesor dictante estuvo presente mientras los estudiantes completaban el cuestionario en un laboratorio informático en el cual cada estudiante utilizaba una computadora personal; en la universidad pública, el profesor envió por el cuestionario por mail luego de explicar como responderlo, durante la primer clase del ciclo lectivo.

El investigador estuvo presente en las clases de programación, conocía la totalidad de la bibliografía y ejercicios que se impartían en las asignaturas.
En la Tabla 1 se presenta el cronograma general de la asignatura a lo largo de las ocho semanas. Cada día comenzó con un breve espacio dedicado a retomar el trabajo de la clase anterior. Si bien no hubo coincidencia absoluta en ambas universidades en relación al contenido detallado y ejercitaciones, hubo coincidencia en los temas referidos en la Tabla 1 así como en el orden asignado.

Tabla 1. Cronograma general de la asignatura

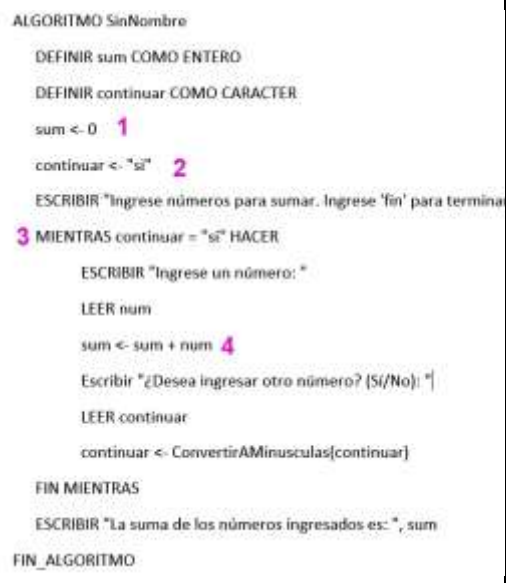
Tema	Contenido general	Tiempo aproximado
Introducción a la Programación	Conceptos iniciales: programación, algoritmo, programa, lógica, paradigmas de programación. Pseudocódigo. Diagramas de flujo y estructurados.	2 clases
Estructuras de control	Estructura secuencial. Variables, constantes, tipos de datos y operadores. Asignación. Inicialización.	1 clase
	Expresiones lógicas. Estructura de bifurcación. Estructuras de bifurcación anidadas y múltiples.	1 clase
	Estructuras de iteración. Estructuras de iteración exactas e inexactas.	2 clases
Algoritmos básicos	Acumulador. Contador. Intercambio de variables. Búsqueda. Ordenamiento.	2 clases

La primer parte de la encuesta solicitó datos demográficos y estudios previos (escuela media de origen y terminalidad, universidad en la cual se cursa actualmente la carrera e institución previa en el caso de estudiantes que cursaron la misma carrera o similar en otra institución antes del ingreso actual. En el post-test esta sección incorporó preguntas de opinión comparativas entre el aprendizaje en escuela media y en la universidad.
La segunda parte de la encuesta contenía tres tipos de preguntas. Los primeros dos tipos de preguntas estaban dirigidas a identificar conceptos en cinco categorías: 1. Inicialización, 2. Bifurcación, 3. Iteración, y algoritmos básicos: 4. Contador, 5. Acumulador. Las categorías se determinaron eligiendo conceptos básicos en programación estructurada que se suelen encontrar en materiales de aprendizaje desarrollados para escuela secundaria, excluyendo los propios de la programación orientada a eventos (como el concepto de concurrencia). Se tuvo en cuenta que fueran categorías lexicalizadas, es decir, que exista una palabra o término breve para identificarlas, y que el término evocara un proceso, no un sustantivo ni una descripción. No se contemplaron conceptos con alto grado de ambigüedad por ser más difíciles de representar (por ejemplo, el concepto de variable, para el cual se debería diferenciar entre variable en matemática o variable en informática).
El primer tipo de preguntas (Ver Tabla 2), era de reconocimiento *procedural* en un fragmento de código. En estos fragmentos sólo se exponían los conceptos básicos 1. a 5. (no se representaron, por ejemplo, arreglos o algoritmos de búsqueda). Algunos fragmentos estaban programados con bloques visuales (se utilizaron bloques en

Scratch), otros fragmentos estaban programados en *pseudocódigo*. Se presentaba un fragmento de código antecedido por la pregunta ¿Cómo funciona este código?, inspirada en Brennan y Resnick [17] también citado en Grover [18]. Este tipo de pregunta induce al estudiante a explicar las instrucciones. A través de sus respuestas, se evalúa la identificación precisa de los conceptos presentes en el código, lo que a su vez influye en la puntuación final de la categoría correspondiente. Por ejemplo, si el estudiante señala la repetición de un conjunto de instrucciones, esto indica una comprensión de cómo funciona la iteración, lo que resultaría en la asignación de un punto a dicha categoría. Es importante destacar que en este tipo de respuestas no es esencial que el estudiante refiera explícitamente a la categoría en cuestión (nombrándola), sino que demuestre comprensión acertada del funcionamiento correspondiente.

Tabla 2. Preguntas de reconocimiento procedural en un fragmento de código

Instrucciones: "En la siguiente sección te presentaremos fragmentos de código programados en bloques visuales que refieren a movimientos u otras acciones que se aplican a un personaje. Verás la pregunta ¿Cómo funciona este código? En los renglones de abajo, explica lo que hace cada instrucción. Para indicar de qué instrucción se trata puedes mencionar los números en color negro para referirte a ellas. Por ejemplo: En la instrucción 1: se realiza xxxxx..., en la 2:..... , 3..... . Si no comprendes alguna instrucción simplemente no la menciones."		Conceptos representados en el fragmento de código:
Ejemplo de pregunta de reconocimiento de conceptos en un fragmento de código *en bloques visuales*	¿Cómo funciona este código? 	*Inicialización*, *Iteración*, *Acumulador*

Ejemplo de pregunta de reconocimiento de conceptos en un fragmento de *pseudocódigo*		*Inicialización, Iteración, Acumulador*

El segundo tipo de preguntas (Ver Tabla 3), evaluaban el conocimiento *declarativo*, eran preguntas de *reconocimiento verbal* de conceptos. Algunas eran preguntas de selección múltiple, por ejemplo, "Recuerdas haber escuchado o leído el término iteración, repetición o bucle?", las respuestas posibles eran "No lo había escuchado ni leído / Lo había escuchado o leído / Lo comprendo poco / Lo comprendo bien / Lo utilicé en mis programas".

Tabla 3. Preguntas de reconocimiento verbal de conceptos

Instrucciones: "A continuación mencionamos algunos conceptos. En algunos casos, puede suceder que no lo conozcas pero que hayas leído o escuchado hablar de ellos en alguna clase o situación. Indicá la opción que corresponde."		Opciones de respuesta
Ejemplo de pregunta de selección múltiple	Inicialización	*No lo había escuchado/leído* *Lo había escuchado/leído* *Lo comprendo un poco* *Lo comprendo bien* *Lo utilicé en mis programas*
	En caso de que conozcas o hayas escuchado hablar del concepto de inicialización, ¿en qué lugar fue?	*En esta asignatura (Post)* *En la escuela secundaria* *En un curso* *Buscando en Internet* *No lo recuerdo*

La codificación de las respuestas, para ambos tipos de preguntas de reconocimiento, se basó en las técnicas de análisis verbal descritas por Chi [16], también utilizadas por Grover [18], diseñadas para analizar datos cualitativos de manera cuantificable, especialmente en disciplinas STEM.
El tercer tipo de preguntas eran abiertas, por ejemplo, "¿Qué conceptos vistos en la escuela has podido poner en práctica o has recordado cuando comenzaste a estudiar programación en la universidad?", o "Qué lenguajes prefieres o te parecen más adecuados para aprender a programar: lenguajes basados en bloques visuales como Scratch o lenguajes basados en texto como C o Python?" "¿Por qué?". Estas preguntas se incluyeron exclusivamente en el Post-test. Esta decisión se fundamentó en el hecho de que dichas preguntas, como puede verse en la Tabla 4, requerían que el estudiante hubiera completado previamente el cursado de ocho semanas de la asignatura. Su propósito principal radicaba en permitir al estudiante comparar y contrastar su aprendizaje actual con el conocimiento adquirido durante su educación secundaria.

Tabla 4. Preguntas abiertas

Comparativas	En tu opinión, ¿en qué aspectos difiere la enseñanza de programación en la escuela secundaria y en la universidad? ¿Consideras que los conocimientos adquiridos en la escuela secundaria te prepararon adecuadamente para los desafíos de la programación en la universidad?
Mejoras sugeridas.	¿Hay algún aspecto específico en el que crees que la enseñanza de programación en la escuela secundaria podría mejorarse para ayudar a los estudiantes a prepararse mejor para la universidad? ¿Qué sugerencias tendrías para mejorar el plan de estudios de programación en la universidad?
Comentarios finales.	¿Hay algo más que te gustaría compartir sobre tu experiencia en la enseñanza de programación en la escuela secundaria y en la universidad?

La Tabla 5 ilustra algunos ejemplos de las respuestas previas y posteriores al cursado para las preguntas de reconocimiento de código y su puntuación en cada caso.

Tabla 5. Análisis de respuestas de reconocimiento de conceptos en código

¿Cómo funciona este código?

Caso	*Respuesta previa al cursado*	*Puntaje*	*Respuesta posterior al cursado*	*Puntaje*
Estudiante 1 (18 años)	"1 Pregunta si toca el color amarillo 2 dice estoy programando 2 segundos 3 Mueve 10 pasos	Bifurcación: 0	"1 Si toca el color amarillo muestra el mensaje que es la 2 Si no lo toca hace la 3 mueve 10 pasos (3) Y espera 0.2 segundos"	Bifurcación: 1
Estudiante 2 (19 años)	"Cuando toque el color amarillo realiza la acción 2"	*Bifurcación: 0.5*	"La instrucción 1 es una condición. Si se cumple que es amarillo hace la acción 2 muestra el mensaje 2 segundos Si no hace la 3 Mover 10 pasos y esperar 0.2"	Bifurcación: 1

Como sugiere Chi en su guía para el análisis de datos verbales, todo mensaje expresado verbalmente, sea en forma oral o escrita, es susceptible de ser codificado en categorías [16]. En este caso, el puntaje asignado a cada categoría dependía de la completitud de la explicación verbal. Por ejemplo, en la tabla anterior, el estudiante 1, en el *pre-test*, describe las acciones secuencialmente incluyendo tanto las que están sujetas a un valor verdadero de la condición, como las que deben efectuarse ante un valor falso, de lo cual se interpreta que no comprende el funcionamiento del condicional, por lo tanto, no se asigna puntaje a la categoría *bifurcación*. Las demás instrucciones (Mostrar mensaje, moverse y esperar) no están clasificadas en ninguna de las categorías a evaluar, por lo tanto, el puntaje es nulo. En contraste, el mismo estudiante, identifica correctamente una bifurcación en el *post-test*, luego de las 8 semanas de cursado, tanto la primer parte como la segunda, con lo cual se asigna un puntaje de 1 a dicha categoría. El estudiante 2, en cambio, sólo identificó correctamente la parte positiva de la bifurcación, pero no explica las acciones en el caso falso, con lo cual se asigna un puntaje de 0.5.

Las preguntas verbales se centraban en la mención específica de un concepto, lo que podría desencadenar la activación o "recordatorio" de conceptos y su funcionamiento

previamente adquiridos. Esta dinámica implicaba el riesgo de anticipar las respuestas a las preguntas de reconocimiento de código si las preguntas verbales se formulaban en primer lugar. Por consiguiente, se decidió posponer la realización de estas preguntas hasta después de las preguntas de reconocimiento de código. En la Tabla 6 se presenta un ejemplo de análisis de preguntas verbales.

Tabla 6. Análisis de respuestas de reconocimiento verbal de conceptos

Instrucciones:

(A) A continuación mencionamos el nombre de algunos conceptos. En algunos casos, puede suceder que no lo conozcas pero que hayas escuchado hablar de ellos en alguna clase. Indicá la opción que corresponde. (Opciones y puntajes: No lo había escuchado (0) / Lo había escuchado (0.25) / Lo comprendo un poco (0.5) / Lo comprendo bien (0.75) / /Lo utilicé en mis programas (1).

(B) En caso de que conozcas o hayas escuchado hablar del concepto de inicialización, ¿en qué lugar fue? (Opciones: *En otra carrera* (PRE), *En esta materia* (POST)*/En la escuela secundaria/En un curso/Buscando en Internet/No recuerdo*

Concepto: *Inicialización*

Caso	Respuesta previa al cursado	Puntaje	Respuesta posterior al cursado	Puntaje
Estudiante 3 (19 años)	A. No lo había escuchado	0	A. Lo comprendo bien	0.75
	B. Sin respuesta		B. En esta materia	
Estudiante 4 (20 años)	A. Lo había escuchado	0.25	A. Lo utilicé en mis programas	
	B. En un curso		B. En esta materia	

Para cada categoría, se presentaron una pregunta verbal y tres fragmentos de código. El puntaje final para cada categoría (inicialización, iteración, bifurcación, contador, acumulador), se obtenía como un promedio del puntaje de las respuestas para cada uno de los items que pertenecían a dicha categoría. Así por ejemplo, en el caso de la categoría *iteración*, se presentaba una pregunta verbal ("Recuerdas haber escuchado o leído alguno de estos términos?: iteración/repetición/bucle/ciclo") y tres fragmentos de código. Luego, el puntaje se obtenía sumando el puntaje de la respuesta a la pregunta verbal más el puntaje de cada fragmento de código y dividiendo esta suma por 4.

4 Resultados parciales

Con respecto a la segunda parte de la encuesta dedicada al conocimiento de conceptos básicos antes y después del curso (Ver Tabla 7 y Figura 1), los estudiantes mostraron menor precisión –con una diferencia significativa- antes del cursado de la asignatura

que luego de las 8 primeras semanas para identificar el concepto de *inicialización* (10% vs. 64%), $t(94) = -16.82$, $p < .001$, así como para los conceptos de *bifurcación* (63% vs. 79%), $t(94) = -5.95$, $p < .001$, *iteración* (46% vs. 91%), $t(94) = -20.21$, $p < .001$, *contador* (46% vs. 90%), $t(94) = -10.5$, $p < .001$, *acumulador* (57% vs. 80%), $t(94) = -7.51$, $p < .001$.

Tabla 7. Conocimiento de conceptos antes y después del cursado

	Inicialización	Iteración	Bifurcación	Contador	Acumulador
Pre-test	0,11	0,55	0,62	0,46	0,58
Post-test	0,66	0,86	0,78	0,89	0,80

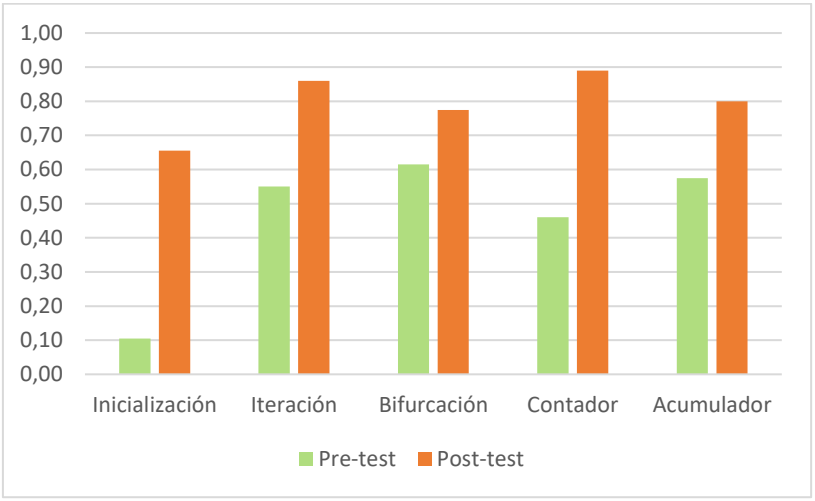


Fig. 1. Conocimiento de conceptos antes y después del cursado

Desglosando el análisis en conocimiento declarativo y procedural (distinguiendo el reconocimiento verbal de conceptos de su identificación en fragmentos de código), se observa un cambio de tendencia entre el *pre-test* y el *post-test*. Tal como se muestra en la Tabla 8, antes del curso, el 16% afirmaba haber escuchado o leído el concepto de inicialización aunque sólo el 5% lograba identificarlo en el código. Por otra parte, sólo el 39% dice haber escuchado o leido el término iteración, pero un mayor número de estudiantes (71%) lograba identificarlo en el código, un 55% había escuchado o leído el término bifurcación y un 71% lo identificaba en el código. Las diferencias verbal procedural no eran tan importantes para el resto de los items.

Tabla 8. Conocimiento verbal y procedural antes y después del cursado

	Inicialización	*Iteración*	*Bifurcación*	*Contador*	*Acumulador*
Verbal	0.16	0.39	0.55	0.43	0.53

Pre-test	*Proced.*	0.05	0.71	0.68	0.49	0.62
Post-test	*Verbal*	0.71	0.83	0.71	0.84	0.83
	Proced.	0.60	0.89	0.84	0.94	0.77

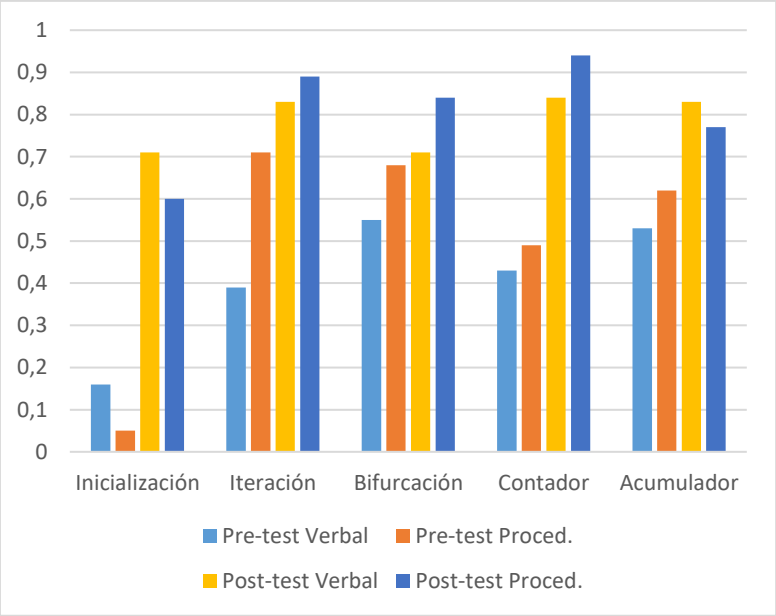


Figura 2. Conocimiento verbal y procedural antes y después del cursado

Sin embargo, luego de la cursada, un 71% identifica verbalmente el concepto de inicialización mientras el 60% lo identifica correctamente en el código, un 83% recuerda el término iteración y un 89% lo identifica en el código, y algo similar ocurre con los demás conceptos. En el post-test las medidas verbal procedural se acercan entre sí, como se ilustra en la Figura 2. Cambia la relación entre el reconocimiento verbal de los conceptos y la comprensión del código. Antes del ingreso, el reconocimiento procedural es superior al recuerdo de los conceptos que subyacen al código. Luego del curso, mejoran ambas medidas.

5 Discusión y conclusiones

Los resultados de este estudio revelan una mejora significativa en la identificación de conceptos básicos luego de la introducción a la asignatura universitaria, en comparación con los conocimientos previos al finalizar la escuela media. Esta mejora, aunque esperada y congruente con las expectativas asociadas a un curso de este tipo, es menos

relevante que la baja comprensión inicial de los conceptos antes del ingreso a la universidad.

Un hallazgo de particular interés y sutileza reside en la relación entre el conocimiento verbal y procedural durante el proceso de adquisición de conocimiento. La diferencia principal entre el pre-test y el post-test con respecto a la interacción entre lo declarativo (el concepto verbalmente retenido) y lo procedural (el proceso cuyo funcionamiento se comprende aunque no se le asigne el término correcto) radica en que, en el pre-test, los estudiantes pueden comprender el código aunque no logren expresar el término conceptual asociado. Sin embargo, en el post-test, esta brecha disminuye considerablemente: los estudiantes identifican mejor tanto el proceso como la terminología correspondiente. En otras palabras, la universidad aporta el conocimiento conceptual que le da sentido a los bloques de código, hecho que se vislumbra en las respuestas de los estudiantes, quienes indican que "en la universidad […] las explicaciones son mejores". Es evidente que los estudiantes realizan prácticas informales que les permiten comprender el funcionamiento de un fragmento de código, aunque no puedan darle un nombre específico (lexicalizarlo). Sin embargo, es importante destacar que el nivel de conocimiento de estas categorías fundamentales es notablemente bajo, especialmente teniendo en cuenta que los estudiantes han optado por cursar la carrera universitaria.

Este fenómeno parece estar en consonancia con las observaciones de algunos docentes, quienes han señalado que, durante los primeros meses de la carrera, los errores más frecuentes están relacionados con la comprensión de cómo funcionan las estructuras básicas del código. Esto se debe a que los estudiantes tienden a estar más familiarizados con la copia de código que con la generación de código propio, tendencia que se evidencia cuando los estudiantes no logran explicar el funcionamiento de un fragmento de código propio. Sin embargo, es importante mencionar, a favor de este incipiente conocimiento previo, que el principal problema no radica necesariamente en la ausencia total de enseñanza previa en programación. Si bien puede ser tentador argumentar que los estudiantes carecen por completo de cualquier conocimiento previo en programación, y repetir la queja habitual de que el sistema educativo previo es responsable de las dificultades en el nivel actual, no debería perderse de vista que algunos estudiantes pueden reconocer procesos simples aunque no los nombren explícitamente. Si este conocimiento previo es aprovechado adecuadamente por los profesores universitarios, existe una esperanza real de mejorar el nivel actual a través de nuevas propuestas didácticas.

Actualmente, observamos una tendencia en entornos virtuales en la que el profesor explica el código mientras lo escribe (en tiempo real). Si bien esto puede resultar útil para enseñar la sintaxis de un lenguaje específico, puede ser menos efectivo al explicar conceptos como algoritmos básicos. En situaciones como estas, es prudente adoptar un enfoque que combine varias estrategias pedagógicas. En lugar de centrarse exclusivamente en el detalle de cada línea de código, se recomienda enfocarse en el proceso global recurriendo a técnicas como el uso de analogías, comparaciones entre diferentes procesos y la utilización de diagramas explicativos (por ejemplo, en PseInt). El objetivo principal es facilitar la asimilación de un patrón típico que pueda ser transferido a cualquier lenguaje de programación, sin quedar limitado a uno específico.

Esta aproximación permite a los estudiantes desarrollar una comprensión más profunda y flexible de los conceptos fundamentales de la programación, lo que les facilita la adaptación a diferentes entornos y tecnologías en el futuro. En cambio, si los estudiantes simplemente se dedican a seguir una a una las instrucciones de código proporcionadas por el profesor y las repiten, corren el riesgo de memorizar la sintaxis sin comprender el proceso subyacente. Posteriormente, podrían intentar aplicar esta sintaxis memorizada a un proceso incorrecto.

Otra estrategia eficaz para los educadores consiste en reforzar verbalmente los conceptos durante las actividades prácticas, explicitando oralmente cuáles son los conceptos subyacentes en la actividad, aunque advertimos que esta no es precisamente la tendencia predominante en las plataformas que permiten la programación en tiempo real.

Sería muy beneficioso para la universidad actual que la identificación de conceptos básicos se haya logrado antes del ingreso de los estudiantes. En respuesta a esta necesidad, se proponen dos caminos no excluyentes. En primer lugar, se sugiere reforzar el conocimiento conceptual en los cursos de ingreso universitarios. En segundo lugar, se recomienda promover estrategias cognitivas para fortalecimiento del conocimiento declarativo (verbal) durante la escuela media y la universidad, con énfasis en la enseñanza de los conceptos fundamentales durante estos períodos formativos.

El conocimiento declarativo/verbal se ubica en un nivel de abstracción superior con respecto a la práctica. En la actualidad, cada estudiante tiene la posibilidad de producir un fragmento de código asistido por IA o copiando y ensamblando desde una variedad de fuentes (GitHub, Stack Overlow, entre otros). Estas prácticas, aunque pueden acelerar su desarrollo profesional futuro, paradójicamente, podrían obstaculizarlo si se emplean prematuramente. Las empresas que emplean asistentes de IA para mejorar su eficiencia probablemente prefieran contratar desarrolladores que posean un sólido conocimiento de los conceptos fundamentales que sustentan el código ya que esto les permitirá generar requerimientos más precisos y detallados para la IA. En este sentido, el riesgo de automatización laboral está inversamente relacionado con la demanda cognitiva de la fuerza de trabajo. Cuanto más elevadas y complejas sean las habilidades de abstracción adquiridas por los estudiantes, menor será el riesgo de ser reemplazados. Éste constituye otro incentivo para seguir investigando en el ámbito de las ciencias cognitivas y su aplicación en el aprendizaje de la ingeniería del software.

5.1 Referencias

1. Stanton, J. *et al.* State of the states landscape report: State-level policies supporting equitable K–12 computer science education. *Education Development Center* (2017).
2. Furber, S. Shut down or restart? The way forward for computing in UK schools. The Royal Society. (2015).
3. Falkner, K. *et al.* An International Comparison of K-12 Computer Science Education Intended and Enacted Curricula. in *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* 1–10 (ACM, Koli Finland, 2019). doi:10.1145/3364510.3364517.

4. Hubwieser, P., Armoni, M., Giannakos, M. N. & Mittermeir, R. T. Perspectives and Visions of Computer Science Education in Primary and Secondary (K-12) Schools. *ACM Trans. Comput. Educ.* **14**, 1–9 (2014).
5. Guo, M. & Ottenbreit-Leftwich, A. Exploring the K-12 computer science curriculum standards in the U.S. in *Proceedings of the 15th Workshop on Primary and Secondary Computing Education* 1–6 (ACM, Virtual Event Germany, 2020). doi:10.1145/3421590.3421594.
6. Keane, N. & McInerney, C. Report on the provision of courses in computer science in upper second level education internationally. *Report commissioned by the National Council for Curriculum and Assessment* (2017).
7. Webb, M. *et al*. Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Educ Inf Technol* **22**, 445–468 (2017).
8. Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A. & Engelhardt, K. *Developing Computational Thinking in Compulsory Education-Implications for Policy and Practice*. https://ideas.repec.org/p/ipt/iptwpa/jrc104188.html (2016).
9. Sadosky, F. Una propuesta para refundar la enseñanza de la computación en las escuelas Argentinas. *Reporte de la Fundación Sadosky* 23 (2016).
10. Devteam, educ ar. NAP de Educación Digital, Programación y Robótica. https://www.educ.ar/recursos/150123/nap-de-educacion-digital-programacion-y-robotica.
11. Rodríguez, J. R. & Cortez, M. La posición de las ciencias de la computación en el diseño curricular para la escuela secundaria argentina. *Electronic Journal of SADIO* **19**, (2020).
12. Rodríguez, J., Cortez, M. M. & Boari, S. Explorando el lugar de las áreas de conocimiento de las ciencias de la computacion en la escuela secundaria argentina: Una revisión sistemática. *Electronic Journal of SADIO (EJS)* **21**, 110–124 (2022).
13. Bentley, B. & Sieben, R. Cognitive load theory: An adjunct to constructivist learning theory not an alternative. *Australian Educational Leader* **41**, 48–51 (2019).
14. Bentley, B., Sieben, R. & Unsworth, P. STEM Education in Australia: Impediments and Solutions in Achieving a STEM-Ready Workforce. *Education Sciences* **12**, 730 (2022).
15. Tang, X., Yin, Y., Lin, Q., Hadad, R. & Zhai, X. Assessing computational thinking: A systematic review of empirical studies. *Computers & Education* **148**, 103798 (2020).
16. Chi, M. T. H. Quantifying Qualitative Analyses of Verbal Data: A Practical Guide. *Journal of the Learning Sciences* **6**, 271–315 (1997).
17. Brennan, K. & Resnick, M. New frameworks for studying and assessing the development of computational thinking. in *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada* vol. 1 25 (2012).
18. Grover, S. Robotics and engineering for middle and high school students to develop computational thinking. in *annual meeting of the American educational research association, New Orleans, LA* (2011).