

Transferencia de conceptos básicos de programación de la escuela media a la universidad

Verónica Sofía D'Angelo ^a & Alejandro Mario Hernández ^b

^a Consejo Nacional de Investigaciones Científicas y Técnicas, Rosario, Argentina.

^b Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Rosario, Argentina.
dangelo@irice-conicet.gov.ar, aleh@fceia.unr.edu.ar

Resumen— El estudio presenta una prueba piloto para evaluar la transferencia de conceptos básicos de programación desde la escuela media a la universidad, basada en las expresiones verbales 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 muestran una mejora significativa en la identificación de conceptos básicos después del curso, aunque inicialmente se observa una baja identificación de estos conceptos. Se destaca la relación entre el conocimiento verbal y procedural, con una brecha que disminuye tras la intervención. Se evidencia una preferencia por 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 pensamiento computacional y evaluación de conceptos de programación.

Palabras clave— algoritmos y estructuras de datos, ciencias de la computación, ciencias cognitivas, programación, conceptos básicos, expresiones verbales.

Recibido: 31 de agosto de 2024. Revisado: 20 de octubre de 2025. Aceptado: 4 de noviembre de 2025.

Transfer of Programming Basics from Middle School to College

Abstract— The study presents a pilot test to evaluate the transfer of basic programming concepts from middle school to university, based on the verbal expressions of incoming students to the Information Systems Engineering program. 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 an initial low identification of these concepts is observed. The study highlights the relationship between verbal and procedural knowledge, with a gap that decreases after the intervention. There is evidence of students' preference for learning programming using text-based languages. 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 provides directions for future research in the field of computational thinking and the assessment of acquired programming concepts.

Keywords— algorithms and data structures, computer science, cognitive sciences, programming, basic concepts, verbal expressions.

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. En la actualidad, se espera que los estudiantes tengan un conocimiento más amplio, incluyendo nociones de programación orientada a objetos, manejo 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, ni trasladar una carga excesiva a los años posteriores. 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 ciencias de la computación 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 ciencias de la computación y las Tecnologías de la Información y la Comunicación (TIC), subrayando que las primeras poseen un carácter más específico, científico y menos utilitario en comparación con las TIC. Además, se insiste en que son las ciencias de la computación las que se procuran promover en la educación secundaria. En segundo lugar, se hace hincapié en la distinción entre las ciencias de la computación y la programación, reconociendo que esta última constituye sólo un componente básico aunque fundamental para las ciencias de la computación. 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 obligaciones 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 preuniversitarios 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 ciencias de la computación 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. Un ejemplo específico es la exploración de la diferencia entre conocimiento declarativo y procedimental.

1.2 Diferencia entre conocimiento declarativo y procedimental

Tradicionalmente, el conocimiento procedimental ha sido subestimado en el ámbito educativo. Cobra fuerza recientemente en la escena pedagógica de la mano de la tecnología y las iniciativas para promover el pensamiento computacional. Si bien las teorías de Bruner y Ausubel sentaron las bases para la comprensión del conocimiento declarativo, se enfocaron en conectar conceptos en lugar de organizar procedimientos. Fue a partir de investigaciones posteriores sobre la memoria [15], que se logró discernir claramente entre conocimiento declarativo y procedimental y la importancia de su integración: que el aprendiz no sólo automatice una práctica sino que pueda explicar en qué consiste.

Un problema común en las asignaturas que dependen en gran medida de la práctica, es la interpretación equívoca de la noción de aprendizaje "activo" o "constructivista", al suponer que se aprende por el mero hecho de realizar acciones físicas, suponiendo que el saber declarativo se activará como consecuencia de la acción, sin evocarlos de manera explícita. En realidad, el aprendizaje activo implica estar mentalmente involucrado, no solo físicamente. El conocimiento declarativo es esencial en las etapas iniciales del aprendizaje de procedimientos. Las personas recuerdan conceptos en función de las actividades mentales (elaboraciones) que realizan durante la etapa de aprendizaje.

Para ilustrar esta relación compleja entre el conocimiento declarativo y el procedimental, podemos compararla con el proceso de aprender a tocar un instrumento musical. Algunas personas talentosas adoptan un enfoque autodidacta,

practicando el instrumento sin necesariamente comprender la teoría musical, memorizando la secuencia completa de una pieza utilizando videos como guía. A medida que el aprendiz alcanza un nivel suficiente de destreza para interpretar la pieza, deja de concentrarse en cada movimiento individual de sus dedos sobre las teclas. De hecho, analizar cada movimiento podría interferir con la fluidez del rendimiento. Sin embargo, si en el futuro el músico desea componer o ampliar su repertorio, necesitará volver su atención a las notas y movimientos, intentando comprender lo que al principio pasó por alto. En este punto, el automatismo adquirido puede convertirse en un obstáculo, ya que el esfuerzo requerido para expandir el conocimiento puede parecer menos gratificante que el disfrute inicial de la práctica.

El ejemplo anterior, aplicado al ámbito de la programación de computadoras, ilustra que los estudiantes que han automatizado habilidades para copiar y ensamblar código o para trabajar con IA antes de comprender los fundamentos, pueden mostrar resistencia a aprenderlos más tarde. Es probable que encuentren dificultades significativas al intentar comprender los conceptos subyacentes, lo que podría desmotivarlos y llevarlos a abandonar el estudio. La automatización no asegura un aprendizaje correcto; de hecho, existe el riesgo de automatizar errores.

1.3 La importancia del lenguaje en la ingeniería del software

La competencia en programación no es la única habilidad relevante en el ámbito de la ingeniería del software; los desarrolladores también deben ser capaces de comunicarse efectivamente [16]. El dominio del lenguaje y la capacidad para comprender y expresar verbalmente los conceptos son fundamentales para el desempeño exitoso en esta disciplina. Estas habilidades son fundamentales para garantizar una comunicación efectiva durante el desarrollo de software, lo que contribuye a una comprensión adecuada de los requisitos del sistema y a mantener una dinámica de equipo fluida y eficiente.

Específicamente en el contexto de la programación, una comprensión clara y precisa de los requisitos del proyecto es esencial para crear soluciones de software efectivas. Los programadores deben ser capaces de interpretar y traducir los requerimientos del cliente en código funcional, lo que requiere una comunicación efectiva tanto con el equipo de desarrollo como con los interesados del proyecto. Esto implica no solo expresar ideas técnicas de forma comprensible, sino también entender y responder a las contribuciones de otros miembros del equipo. En este sentido, los programadores no solo utilizan lenguajes de programación para crear sus programas, sino que también recurren al lenguaje natural para describir su funcionamiento. Esto lo hacen también cuando utilizan pseudocódigo, que sirve como una representación intermedia entre el lenguaje natural y el código de programación, así como cuando añaden comentarios en el código, elementos esenciales para documentar y mejorar la claridad del programa.

Actualmente, ante el rumor extendido de que la inteligencia artificial eventualmente reemplazará a los programadores,

suponiendo que este escenario se materialice, surgirá la necesidad de contar con desarrolladores capaces de expresar verbalmente requisitos precisos para que la inteligencia artificial pueda programar en consecuencia. Esto implica la formación de una fuerza laboral que opere en niveles de abstracción cada vez más altos, capaz de colaborar con una inteligencia artificial que se encargue de desarrollar los requisitos de nivel inferior. Una vez más, se destaca la importancia del lenguaje como herramienta fundamental en este proceso.

Varios estudios previos que han investigado los beneficios de la apropiación del lenguaje académico demuestran que ciertos tipos de conversaciones y el uso de un lenguaje específico son cruciales para el aprendizaje con comprensión en diversas disciplinas, incluyendo matemáticas y ciencias [17]-[20]. Además, se ha comprobado el papel fundamental que realizan los profesores al “traducir” el “lenguaje de los estudiantes” al lenguaje académico, facilitando así la apropiación de los estudiantes de las tareas académicas y su integración en el discurso intelectual; estos autores mostraron que el habla de los profesores juega un papel mucho más importante en el aprendizaje de los estudiantes de lo que se piensa [21], [22]. Las discusiones estructuradas y significativas en el aula son mecanismos primarios para promover una comprensión profunda de conceptos complejos y un razonamiento sólido [23].

Dado que el lenguaje desempeña un papel crucial en la adquisición de conocimientos y la comprensión profunda en ciencias y programación, algunos autores destacan la necesidad de enfocarse en el desarrollo del lenguaje específico de ciencias de la computación, como un componente esencial que conecte los conceptos fundamentales [17], [24]. Al igual que en la enseñanza de las ciencias, donde diferentes formas de hablar en clase sobre conceptos conducen a la interiorización de los mismos, en la enseñanza del pensamiento computacional necesitamos utilizar el habla de manera efectiva para ayudar a los estudiantes a interiorizar los conceptos difíciles que enfrentan, por ejemplo, en programación.

1.4 El lenguaje en la evaluación del conocimiento básico en programación

El lenguaje facilita la adquisición de conceptos y también la evaluación de dicha adquisición. Grover [24] inició investigaciones para el diseño de métodos adecuados de evaluación del pensamiento computacional. Su premisa básica era que, como el lenguaje está presente en toda adquisición de conocimiento, al ingresar en un campo disciplinar, lo primero que se adquieren son los términos propios de ese campo, es decir, el vocabulario específico. De manera tal que el dominio del vocabulario específico del pensamiento computacional puede ser considerado un indicador de dominio básico al inicio de una asignatura introductoria.

En 2011, Grover propuso centrarse en descripciones verbales proporcionadas por los estudiantes durante una intervención educativa sobre pensamiento computacional en

robótica. Encontraron que los estudiantes mencionaron una mayor cantidad de términos técnicos con posterioridad a la intervención y sus explicaciones tenían relación directa con el trabajo desarrollado [24]. En 2019, Grover [25] presenta un ejemplo de evaluación innovadora que prioriza los conceptos sobre la codificación, es decir, a diferencia de otras evaluaciones basadas en la resolución de problemas algorítmicos, evalúa en primer lugar el conocimiento de conceptos básicos introductorios (como variables, expresiones y estructuras de control) sin estar ligados a la sintaxis de un lenguaje particular (aunque se utilizan bloques visuales de Scratch). De este modo, es posible detectar si el estudiante comprendió las estructuras de control que integran un algoritmo aunque no haya llegado a obtener una solución global correcta. Esta evaluación se amplió y sistematizó de manera modular [26] para ser generalizada a diferentes idiomas, creando patrones de diseño reutilizables y adaptables según el contexto, focalizados en el aprendizaje de estructuras básicas (variables, expresiones, bucles, condicionales y abstracciones).

Siguiendo a Chi [30] y Grover [24], propusimos evaluar el conocimiento de conceptos de programación que forman parte de fragmentos de código, a través de la expresión verbal de los estudiantes para identificar dichos conceptos, durante los primeros dos meses de una asignatura anual. En dicho período se espera una adquisición de términos específicos aunque no se haya desarrollado la destreza para resolver problemas algorítmicos en su totalidad.

1.5 Objetivos

El objetivo de esta investigación, que se circunscribe al área de programación y pone especial énfasis en el papel del lenguaje en la formación computacional, fue diseñar una prueba piloto basada en las expresiones verbales de los estudiantes para evaluar el nivel de transferencia de conceptos básicos de programación en la transición desde la escuela media a la universidad. Es importante distinguir que no se pretende evaluar la efectividad de un curso universitario de introducción a la programación sino los conocimientos previos de los estudiantes provenientes de escuelas medias. Dichos conceptos se espera sean transmitidos en las primeras ocho semanas del curso.

Para cumplir dicho objetivo, 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.

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 eran 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.

2 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 [27], es pertinente destacar que los estudios previos han tendido a enfocarse en la efectividad de intervenciones pedagógicas únicas en grupos específicos de estudiantes. Además, se evaluó principalmente la resolución de problemas algorítmicos completos sin indagar en la comprensión pormenorizada de cada una de las estructuras de control o estructuras de datos.

En contraste con estas investigaciones anteriores, el presente estudio se propone analizar en detalle las nociones básicas sobre programación que los estudiantes pueden haber adquirido en escuelas secundarias con diferente modalidad (técnica, ciencias naturales, ciencias sociales). Para ello, se analizará el desempeño de 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 contextos educativos, con una heterogeneidad significativa en cuanto a la carga horaria dedicada a informática y programación. El único punto en común que se presume 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 ocho semanas de la asignatura de programación, se realizó una encuesta antes y otra después del cursado de dicha asignatura.

2.1 Participantes

Una muestra inicial de 172 estudiantes de primer año de Ingeniería Informática fue invitada a participar voluntariamente de una encuesta de respuesta domiciliaria que consistió en dos test. Un test previo al cursado de la asignatura y otro test a los dos meses de cursado.

Se descartaron los datos de veintinueve (29) estudiantes por no haber respondido la encuesta inicial (*pretest*) y los de otros once (11) estudiantes porque, habiendo respondido la encuesta inicial, no respondieron la segunda parte (*posttest*). Por otro lado, se descartaron dos (2) casos de estudiantes que habían concurrido a un colegio pre universitario técnico (con un diseño curricular autónomo que difiere en gran medida de la currícula tradicional), y treinta y seis (36) casos de estudiantes que recursaban la asignatura en la misma universidad o en cursados previos de la misma carrera en otras universidades.

Una muestra definitiva de noventa y cuatro (94) 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).

2.2 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 cómo responderlo, durante la primera 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 las ejercitaciones, hubo coincidencia en los temas referidos en la Tabla 1, así como en el orden asignado. Dichos temas y conceptos son los que se evalúan en la encuesta.

La primera 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 *postest* 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.

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

Fuente: Los autores.

Tabla 2
Preguntas de reconocimiento procedural en un fragmento de código

Instrucciones:
“En la siguiente sección te presentaremos fragmentos de código [...] En los renglones de abajo, explica lo que hace cada instrucción. [...]”

Ejemplo de pregunta de reconocimiento de conceptos en bloques visuales (Inicialización, Iteración, Acumulador)

¿Cómo funciona este código?

Ejemplo de pregunta de reconocimiento de conceptos en pseudocódigo (Inicialización, Iteración, Acumulador)

```

ALGORITMO SinNombre
  DEFINIR sum COMO ENTERO
  DEFINIR continuar COMO CARACTER
  sum <- 0 1
  continuar <- "si"
  ESCRIBIR "Ingrese números para sumar. Ingrese 'fin' para terminar."
  2 MIENTRAS continuar = "si" HACER
    ESCRIBIR "Ingrese un número: "
    LEER num
    sum <- sum + num 3
    ESCRIBIR "¿Desea ingresar otro número? (Si/No): "
    LEER continuar
    continuar <- ConvertirAMinúsculas(continuar)
  FIN MIENTRAS
  ESCRIBIR "La suma de los números ingresados es: ", sum
  FIN ALGORITMO
    
```

Fuente: Adaptado de [24], [28].

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 [28] también citado en Grover [24]. 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. 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. Indica la opción que corresponde.”	
Opciones de respuesta	
<i>Inicialización</i>	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

Fuente: Los autores.

La codificación de las respuestas, para ambos tipos de preguntas de reconocimiento, se basó en las técnicas de análisis verbal descritas por Chi [30], también utilizadas por Grover [24], 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 *postest*. 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?

Fuente: Los autores.

Tabla 5
Análisis de respuestas de reconocimiento de conceptos en código
¿Cómo funciona este código?

Caso		Puntaje
Estudiante 1 (18 años)	<i>Respuesta previa al cursado</i> “1 Pregunta si toca el color amarillo 2 dice estoy programando 2 segundos 3 Mueve 10 pasos 4 Espera 0,2 segundos” <i>Respuesta posterior al cursado</i>	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)	<i>Respuesta previa al cursado</i> “Cuando toque el color amarillo realiza la acción 2” <i>Respuesta posterior al cursado</i>	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

Fuente: Los autores.

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.

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 [30]. 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 *pretest*, 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 *postest*, 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. Indica 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: <i>En otra carrera</i> (PRE), <i>En esta materia</i> (POST)/ <i>En la escuela secundaria</i> /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 B. Sin respuesta	0	A. Lo comprendo bien B. En esta materia	0,75
Estudiante 4 (20 años)	A. Lo había escuchado B. En un curso	0,25	A. Lo utilicé en mis programas B. En esta materia	1

La tabla 7 y la Fig. 1 presentan los resultados generales, que muestran una diferencia a favor del *postest*.

Tabla 7
Conocimiento de conceptos antes y después del cursado

	Inicialización.	Iteración	Bifurcación	Contador	Acumulador
<i>Pretest</i>	0,11	0,55	0,62	0,46	0,58
<i>Postest</i>	0,66	0,86	0,78	0,89	0,80

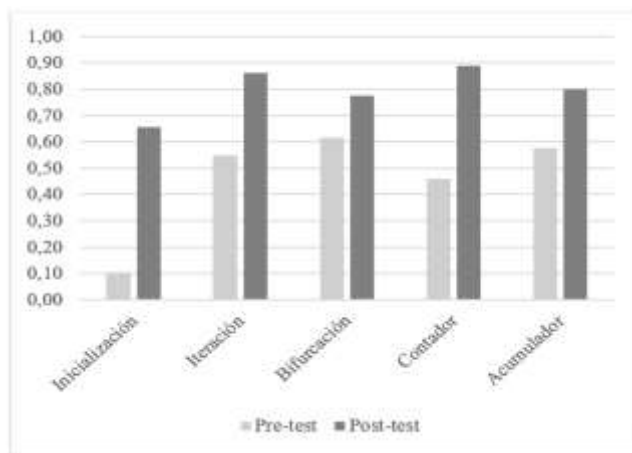


Figura 1. Conocimiento de conceptos antes y después del cursado
Fuente: Los autores

Sin embargo, al profundizar en el análisis y desglosar los resultados en conocimiento declarativo y procedimental — distinguiendo entre el reconocimiento verbal de conceptos y su identificación en fragmentos de código—, se observa un cambio en la tendencia entre el *pretest* y el *postest*.

Tal como se muestra en la Tabla 8 y la Fig. 2, los estudiantes identificaban con mayor facilidad los fragmentos de código que las expresiones verbales. Antes del curso, sólo el 39 % dice haber escuchado o leído el término iteración, aunque un mayor número de estudiantes (71 %) lograba identificarlo en el código, y sólo un 55 % había escuchado o leído el término bifurcación, sin embargo, un mayor número (71 %) podía identificarlo en el código. Las diferencias verbal procedimental no fueron importantes para el resto de los ítems en el *pretest*.

Tabla 8
Conocimiento verbal y procedimental antes y después del cursado

		Inicialización	Iteración	Bifurcación	Contador	Acumulador
<i>Pretest</i>	Verbal	0,16	0,39	0,55	0,43	0,53
	Procedural	0,05	0,71	0,68	0,49	0,62
<i>Postest</i>	Verbal	0,71	0,83	0,78	0,84	0,83
	Procedural	0,60	0,89	0,84	0,94	0,77

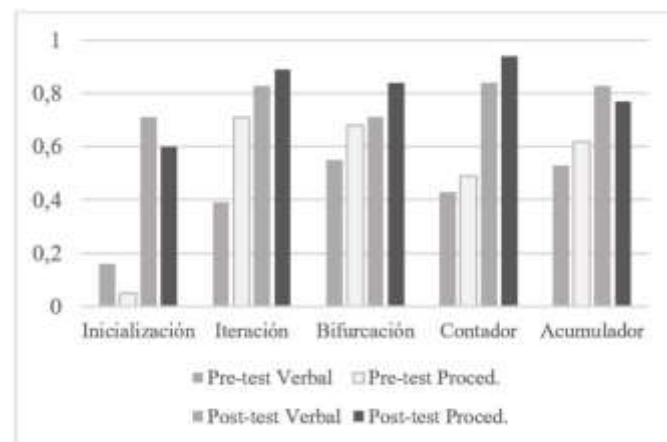


Figura 2. Conocimiento verbal y procedimental antes y después del cursado
Fuente: Los autores

Sin embargo, luego de la cursada, un 83 % recuerda el término iteración y un 89 % lo identifica en el código, un 78 % recuerda el término bifurcación y un 84 % lo identifica en el código. En otras palabras, la brecha entre verbal y procedural se cierra, específicamente para esos conceptos. Esto podría interpretarse como que en el *postest* las medidas verbal procedural se acercan entre sí, como se ilustra en la Fig. 2., lo que podría indicar un cambio en 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.

4 Discusión y conclusiones

Los métodos de evaluación típicos en la enseñanza de programación, ponderan principalmente si el estudiante resolvió correctamente o no el algoritmo. Posiblemente, desarrollando métodos de evaluación que analicen el uso del lenguaje, se puede poner el foco en la comprensión profunda de cada concepto. Esto es importante porque probablemente, aunque los estudiantes no logren resolver exitosamente un problema, muestren en su uso del lenguaje, que tienen cierta comprensión de los conceptos que utilizó para resolverlo.

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 no sorprende, es natural en un curso introductorio. Sin embargo, no es el objetivo de esta investigación averiguar si un curso introductorio es efectivo, sino indagar qué conceptos previos son expresables verbalmente en contraste con el conocimiento procedural y el comportamiento automatizado de los futuros programadores. En ese sentido, es de suma relevancia, la diferencia entre la baja utilización verbal de los términos en el pretest y su utilización en el postest.

Se necesita un estudio más profundo para indagar en esta relación entre el conocimiento verbal y procedural durante el proceso de adquisición de conocimiento. La diferencia principal entre el *pretest* y el *postest* 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 *pretest*, los estudiantes pueden comprender el código aunque no logren expresar el término conceptual asociado. Sin embargo, en el *postest*, esta brecha disminuye considerablemente: los estudiantes identifican mejor tanto el proceso como la terminología correspondiente. En otras palabras, la universidad aporta el conocimiento conceptual verbal que le da sentido y "palabras" a los bloques de código. Este fenómeno se manifiesta también en las respuestas de tipo cualitativo. Los estudiantes expresan que "en la universidad [...] las explicaciones son mejores".

Puede vislumbrarse que los estudiantes, previamente al ingreso universitario, tienen la posibilidad de realizar diversas prácticas informales de escritura y ensamblaje de código que les permiten comprender el funcionamiento de un fragmento de

código, aunque no conozcan ni puedan darle el nombre correcto (lexicalizarlo), a pesar de lo elemental del término.

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 son incapaces de 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 parece provenir necesariamente de 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 nivel 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.

Una estrategia eficaz para los educadores consistiría en reforzar verbalmente los conceptos durante las actividades prácticas, explicitando oralmente y con detenimiento 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. Actualmente, observamos una tendencia en entornos virtuales y en canales de video, a que el profesor explique el código mientras lo escribe rápidamente, y los estudiantes se dedican a seguir una a una las instrucciones de código proporcionadas por el profesor. En esa repetición se corre el riesgo de memorizar la sintaxis sin comprender el proceso subyacente. Posteriormente, podrían intentar aplicar esta sintaxis memorizada a un proceso incorrecto.

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, el estudiante puede producir un fragmento de código asistido por IA o copiando y ensamblando desde una variedad de fuentes (GitHub, Stack Overflow, 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 desarrollando sistemas de evaluación e investigaciones en el ámbito de las ciencias cognitivas y su aplicación en la educación para la ingeniería del software.

Referencias

- [1] J. Stanton *et al.*, «State of the states landscape report: State-level policies supporting equitable K–12 computer science education», *Education Development Center*, 2017. Disponible en: <https://www.ecs.org/wp-content/uploads/MassCAN-Full-Report-v10.pdf>.
- [2] S. Furber, «Shut down or restart? The way forward for computing in UK schools», London, 2012. Accedido: 19 de abril de 2025. [En línea]. Disponible en: <https://royalsocietypublishing.org/news-resources/projects/computing-in-schools/report/>.
- [3] K. Falkner *et al.*, «An International Comparison of K-12 Computer Science Education Intended and Enacted Curricula», *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, Koli Finland: ACM, nov. 2019, pp. 1-10. doi: <https://doi.org/10.1145/3364510.3364517>
- [4] P. Hubwieser, M. Armoni, M. N. Giannakos, y R. T. Mittermeir, «Perspectives and Visions of Computer Science Education in Primary and Secondary (K-12) Schools», *ACM Trans. Comput. Educ.*, vol. 14, n° 2, pp. 1-9, jun. 2014, doi: <https://doi.org/10.1145/2602482>.
- [5] M. Guo y A. Ottenbreit-Leftwich, «Exploring the K-12 computer science curriculum standards in the U.S.», en *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*, Virtual Event Germany: ACM, Oct. 2020, pp. 1-6. doi: <https://doi.org/10.1145/3421590.3421594>
- [6] N. Keane y C. McInerney, «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] M. Webb *et al.*, «Computer science in K-12 school curricula of the 21st century: Why, what and when?», *Educ Inf Technol*, vol. 22, n° 2, pp. 445-468, mar. 2017, doi: <https://doi.org/10.1007/s10639-016-9493-x>
- [8] S. Bocconi, A. Chiocciariello, G. Dettori, A. Ferrari, y K. Engelhardt, «Developing computational thinking in compulsory education- Implications for policy and practice», Joint Research Centre (Seville site), 2016. <https://doi.org/10.21125/edulearn.2016.2136>
- [9] F. Sadosky, «Una propuesta para refundar la enseñanza de la computación en las escuelas Argentinas», *Reporte de la Fundación Sadosky*, p. 23, 2016.
- [10] educ ar portal, (6 de abril de 2024). «NAP de Educación Digital, Programación y Robótica». <https://www.educ.ar/recursos/150123/nap-de-educacion-digital-programacion-y-robotica>
- [11] J. R. Rodríguez y M. Cortez, «La posición de las ciencias de la computación en el diseño curricular para la escuela secundaria argentina», *Electronic Journal of SADIO*, vol. 19, 2020, Accedido: 26 de marzo de 2024. [En línea]. Disponible en: <https://sedici.unlp.edu.ar/handle/10915/135055>
- [12] J. Rodríguez, M. M. Cortez, y S. Boari, «Explorando el lugar de las áreas de conocimiento de las ciencias de la computación en la escuela secundaria argentina: Una revisión sistemática», *Electronic Journal of SADIO (EJS)*, vol. 21, n° 2, pp. 110-124, 2022.
- [13] B. Bentley y R. Sieben, «Cognitive load theory: An adjunct to constructivist learning theory not an alternative», *Australian Educational Leader*, vol. 41, n° 1, pp. 48-51, 2019.
- [14] B. Bentley, R. Sieben, y P. Unsworth, «STEM Education in Australia: Impediments and Solutions in Achieving a STEM-Ready Workforce», *Education Sciences*, vol. 12, n° 10, oct. 2022, doi: <https://doi.org/10.3390/educsci12100730>
- [15] J. Aparicio y M. Rodríguez-Moneo, *El aprendizaje humano y la memoria. Una visión integrada y su correlato neurofisiológico*. Madrid: Pirámide, 2015.
- [16] M. Daneva, A. Herrmann, N. Condori-Fernandez, y C. Wang, «Understanding the Most In-demand Soft Skills in Requirements Engineering Practice: Insights from Two Focus Groups», en *Proceedings of the Evaluation and Assessment on Software Engineering*, Copenhagen Denmark: ACM, abr. 2019, pp. 284-290. doi: <https://doi.org/10.1145/3319008.3319352>
- [17] G. H. L. Fletcher y J. J. Lu, «Education Human computing skills: rethinking the K-12 experience», *Commun. ACM*, vol. 52, n° 2, pp. 23-25, feb. 2009, doi: <https://doi.org/10.1145/1461928.1461938>
- [18] J. L. Weber y J. Parham-Mocello, «Staying Consistent: Discipline-Specific Language Used in a Well-Known AP CS A Curriculum», en 2023 IEEE Frontiers in Education Conference (FIE), IEEE, 2023, pp. 1-7. <https://doi.org/10.1109/fie58773.2023.10343026>
- [19] W. Roth, «Thinking with Hands, Eyes, and Signs: MultiModal Science Talk in a Grade 6/7 Unit on Simple Machines», *Interactive Learning Environments*, vol. 4, n° 2, pp. 170-187, ene. 1994, doi: <https://doi.org/10.1080/1049482940040204>
- [20] J. L. Lemke, *Talking science: Language, learning, and values*. ERIC, 1990. Accedido: 27 de marzo de 2024. [En línea]. Disponible en: <https://eric.ed.gov/?id=ED362379>
- [21] Mary Catherine O'Connor Faculty y Sarah Michaels, Chairman Director, «Aligning Academic Task and Participation Status through Voicing: Analysis of a Classroom Discourse Strategy», *Anthropology & Edu Quarterly*, vol. 24, n° 4, pp. 318-335, dic. 1993, doi: <https://doi.org/10.1525/aeq.1993.24.4.04x0063k>
- [22] L. L. Khisty y K. B. Chval, «Pedagogic discourse and equity in mathematics: When teachers' talk matters», *Mathematics education research journal*, vol. 14, n.o 3, pp. 154-168, 2002.
- [23] L. B. Resnick, S. Michaels, y C. O'Connor, «How (well structured) talk builds the mind», *Innovations in educational psychology: Perspectives on learning, teaching and human development*, pp. 163-194, 2010.
- [24] S. Grover, «Robotics and engineering for middle and high school students to develop computational thinking», en annual meeting of the American educational research association, New Orleans, LA, 2011. Accedido: 2 de marzo de 2024. [En línea]. Disponible en: <https://www.shuchigrover.com/wp-content/uploads/2021/03/AERA2011-Shuchi-Grover-Robotics-and-Engineering-for-Middle-and-High-School-Students-to-Develop-Computational-Thinking.pdf>
- [25] S. Grover, N. Jackiw, y P. Lundh, «Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school», *Computer Science Education*, vol. 29, n.o 2-3, pp. 106-135, jul. 2019, doi: <https://doi.org/10.1080/08993408.2019.1568955>
- [26] S. Grover, «Designing an Assessment for Introductory Programming Concepts in Middle School Computer Science», en *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, Portland OR USA: ACM, feb. 2020, pp. 678-684. doi: <https://doi.org/10.1145/3328778.3366896>
- [27] X. Tang, Y. Yin, Q. Lin, R. Hadad, y X. Zhai, «Assessing computational thinking: A systematic review of empirical studies», *Computers & Education*, vol. 148, p. 103798, abr. 2020, doi: <https://doi.org/10.1016/j.compedu.2019.103798>
- [28] K. Brennan y M. Resnick, «New frameworks for studying and assessing the development of computational thinking», en *Proceedings of the 2012 annual meeting of the American educational research association*, Vancouver, Canada, 2012, p. 25. Accedido: 2 de marzo de 2024.

2024. [En línea]. Disponible en:

<http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf>

- [29] S. Grover, R. Pea y S. Cooper, «Designing for deeper learning in a blended computer science course for middle school students», *Computer Science Education*, vol. 25, n° 2, pp. 199-237, abr. 2015, doi: <https://doi.org/10.1080/08993408.2015.1033142>
- [30] M. T. H. Chi, «Quantifying Qualitative Analyses of Verbal Data: A Practical Guide», *Journal of the Learning Sciences*, vol. 6, n° 3, pp. 271-315, jul. 1997, doi: https://doi.org/10.1207/s15327809jls0603_1

V. D'Angelo es Doctora en Psicología por la Universidad Nacional de Córdoba (UNC) y magíster en Psicología Cognitiva del Aprendizaje por la Facultad Latinoamericana de Ciencias Sociales (FLACSO) y la Universidad Autónoma de Madrid (UAM). Actualmente se desempeña como docente e investigadora en la Universidad Abierta Interamericana y como becaria postdoctoral del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), con funciones en el Instituto Rosario de Investigaciones en Ciencias de la Educación (IRICE), integrando el equipo de Desarrollo Cognitivo.

A. M. Hernández docente de la Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Rosario, Argentina.