Telefónica
FUNDACIÓN

La enseñanza de Lenguajes de Programación en la Escuela: ¿Por qué hay que prestarle atención?

Autores

Rodrigo Fábrega Lacoa, PhD
Jorge Fábrega Lacoa, PhD
Alissa Blair, PhD

*Rodrigo Fábrega Lacoa es PhD Educational Theory and Policy The Pennsylvania State University y Gerente de Ucorp S.A; Jorge Fábrega Lacoa, en PhD en Políticas Públicas de la Universidad de Chicago y Profesor en la UDD en el Centro de Investigación de Complejidad Social; Alissa Blair, PhD en Curriculum and Instruction, University of Wisconsin, fellow en el Center of Educational Research.

Fundación Telefónica desarrolla desde hace una década cursos gratuitos de programación y robótica para estudiantes y profesores de todas las edades y niveles académicos, experiencia que ha demostrado positivos resultados en términos pedagógicos y nos ha permitido ser testigos de la capacidad transformadora que tiene la programación en la educación.

Basados en nuestra experiencia y observando las tendencias mundiales, queremos colaborar a la discusión y formación de una política pública en la enseñanza de la programación en nuestro país. Cómo parte de este esfuerzo, durante el año 2016 desarrollaremos 4 estudios: "¿Por qué hay que prestarle atención a la enseñanza de la programación?"; "¿Qué están haciendo otros países?"; "¿Cuáles son las oportunidades curriculares en Chile?", y finalmente, "Bases para la implementación de la Enseñanza de la Programación en el Sistema Escolar".

Mayo 2016

INTRODUCCIÓN

"Se necesitan 500.000 personas que sepan programar. Si Ud. sabe hablarle a un computador, entonces tiene una excelente oportunidad de trabajo con gran futuro", decía la Revista Popular Science de circulación en Estado Unidos en enero de 1965[1]. Esta falta de programadores parece no haber cambiado en las décadas siguientes. De hecho, las referencias a que se requieren más programadores ha sido frecuente desde entonces, y la demanda crece más rápido que la oferta. Por ejemplo, en EE.UU. se estima que para el año 2020 un millón de vacantes laborales ofrecidas por empresas de tecnología simplemente no se llenarán. Mientras que es común ver a la programación como una profesión con una buena expectativa laboral, menos atención ha tenido el hecho de que aprender a programar puede generar una importante transformación en la calidad de la educación.

Wanted:
500,000 Men to

By Stanley L. Englebardt

IF YOU knew how to "talk to computers," chances are you've got it made. If you don't, you may be missing out on a great job opportunity.

People who talk to computers are called programmers. They instruct data-processing machines on how to perform specific jobs. Today there are about 40,000 of these specialists at work. In six years, experts say, 500,000 more will be needed. Many will require a bachelor's, master's, or even doctor's degree. But close to 50 percent will move into this new profession with only high-school diplomas.

Here's why there's such a tremendous demand for programmers.

Computers are really very stupid multimillion-dollar collections of wires and transistors. Plug one in and it does nothing. Yell at it, curse, kick it—and still it remains mute. The reason: no instructions.

But once *people* write instructions,

The computer programer's daily work

You don't have to be a college man to get a good job in computer programing—today even high-school grads are stepping into excellent jobs with big futures

Feed Computers

the computer becomes a marvelous tool. It can tell the exact moment at which an astronaut should fire his retrorockets, or identify an obscure disease and prescribe a course of treatment. It can keep watch over huge inventories and write reorders when the stock gets low. Computers can prepare your paycheck, update accounts-receivable files—even print out past-due notices when you're late in paying bills.

Thousands of new computers are installed each year to do these jobs. Each one must be programed before it can start processing. This means anywhere from 1 to 100 people sitting down to figure out every possible step in a particular operation. These steps are translated into machine language, punched into cards, and fed into the computer. There they are stored for use during the solution of a problem.

Do you have what it takes to be a programmer?

Education is important, but most important is a quick mind, with the ability

En el último tiempo ha resurgido el interés por la enseñanza y aprendizaje de lenguajes computacionales en las escuelas. Se trata de un fenómeno planetario y de rápida penetración en la población. La masificación de Internet, las redes sociales digitales, la telefonía móvil y la creciente importancia de las ciencias computacionales han motivado -luego de 50 años- a una nueva perspectiva para entender el rol de la programación en la educación. Hoy existe una mayor conciencia en torno a que introducir la enseñanza de lenguajes de programación desde la escuela genera impactos cognitivos, facilita el desarrollo de habilidades de resolución de tareas, promueve el pensamiento lógico y, en términos generales, empodera a los estudiantes en el proceso de aprendizaje. Programar ha pasado rápidamente de un tema periférico de especialistas a ser considerado masivamente como una habilidad fundamental que los nativos digitales deben dominar en algún nivel.

Si hace dos décadas, las preocupaciones respecto de las brechas digitales se focalizaban en el acceso o no a Internet, hoy las brechas se relacionan con la capacidad –o la incapacidad- de organizar información en forma de aplicaciones, visualizaciones y algoritmos de diversa índole, ya que como se advierte en el Reporte Final de la Dirección General de Empresa e Industria de la Comisión Europea sobre las e-habilidades "enfocarse en la alfabetización digital [en desmedro de la educación en informática], tiene el riesgo de que la sociedad tendrá una formación como consumidor de tecnologías, pero incapaz de construir y desarrollar nuevas tecnologías"[2].

En este texto revisaremos los orígenes de este interés por la enseñanza y el aprendizaje de la programación en las escuelas. El documento está dividido en cuatro secciones. En la primera sección, ofrecemos una contextualización de la demanda actual por aprender lenguajes de programación. En la segunda, revisaremos la historia de la enseñanza de la programación y las bases conceptuales que se han utilizado para fundamentar la importancia del aprender a programar como parte del currículum escolar. En la tercera sección estudiaremos la evidencia empírica sobre el impacto del aprendizaje de la programación en el desarrollo cognitivo de los estudiantes. En la sección final presentaremos las principales conclusiones y desafíos pendientes en la materia.

[1] Popular Science, January 1965. p. 106-109

[2] European Comission, DG Enterprise and Industry (2014). "e-SKILLS: THE INTERNATIONAL DIMENSION AND THE IMPACT OF GLOBALISATION".

1.- La demanda por habilidades de programación.

En los últimos años se ha renovado el interés por la enseñanza de la programación en las escuelas. Ello se debe a que nos hemos introducido de lleno en una era en que la capacidad de procesar información es una habilidad central. En tal contexto, quien sabe programar está en una situación ventajosa. En poco más de dos generaciones, los computadores pasaron de ser grandes aparatos reservados para algunas empresas tecnológicas a pequeños dispositivos de diferentes tipos, que se encuentran al alcance de la mano de cualquier persona. La masificación de los computadores globalizó la era de la información y abrió para todos las puertas a una fuente inagotable de conocimiento.

Thomas Friedman en su libro La Tierra es Plana (2005), habla de distintas fuerzas que habrían hecho realidad la globalización, una de ellas fue que el 9 de agosto del año 1995 entró en funcionamiento el primer navegador para hacer uso masivo de Internet: Netscape. Así quedaron atrás los días en que había que escribir pequeños códigos en protocolo Gopher para acceder a distintas bases de datos de universidades y centros de investigación. Sólo bastaba un click para navegar en la World Wide Web. El problema pasó desde ¿cómo adquirir conocimiento? a ¿dónde encontrarlo?, y el desarrollo de navegadores hizo posible entrar al universo de datos de manera eficiente. Posteriormente, la creación de redes sociales vía Internet amplió el intercambio directo de información entre personas, rompiendo la unidireccionalidad de la comunicación desde y hacia los medios de comunicación masivo que existía hasta entonces. Compañías como Facebook o Twitter se convirtieron rápidamente en fenómenos de alcance mundial, ampliando así las redes de comunicación entre los ciudadanos, entre estos con sus autoridades y todo tipo de organizaciones en su entorno. Una vez masificado y complejizado el flujo de información, el siguiente cambio vino de la mano de los celulares inteligentes, que hicieron que todo ese flujo de información e intercambio se hiciera móvil. De modo que estar conectado dejó de ser una actividad acotada a los horarios de oficina o que requiriese estar frente a un computador de escritorio y pasó a ser una situación permanente. En suma, la era de la información se cristalizó en la vida cotidiana de las personas y aumentó la complejidad e inmediatez de sus interacciones con otros. Todos estos aumentos en escala e intensidad en el flujo de información han expuesto a la población a una avalancha de datos que es imposible de procesar completamente. Por ello, es natural que haya surgido una demanda por métodos que permitan transformar esos datos en significado y no mero ruido.

Nos hemos introducido de lleno en una era en que la capacidad de procesar información es una habilidad central. En tal contexto, quien sabe programar está en una situación ventajosa.

Aprender a programar se ha convertido en una tendencia mundial. Algunos países han dado el paso de enseñar ciencias computacionales en las escuelas.

Quienes saben programar han sido vitales para generar esos métodos. Ellos han creado múltiples soluciones para satisfacer la demanda por una mejor organización de la información. Ejemplos sobran: motores de búsqueda en Internet, sistemas de compra/venta y recomendación en línea de todo tipo de productos, aplicaciones para celulares que indican dónde viene el bus, cómo llegar a un lugar o cuántos taxis hay disponibles cerca, etcétera. Para cada posible problema ha surgido una o muchas aplicaciones (App) descargables en el celular.

Se estima que en 2016 se producirán 211 mil millones de descargas de aplicaciones para teléfonos celulares en el mundo (www.statista.com). De menor desarrollo todavía, pero en proceso de expansión, se están creando sistemas que ya no comunican a las personas entre sí, sino que conectan a las cosas entre sí: refrigeradores que avisan a la tienda cuando falta comida, relojes de pulsera que envían indicadores de salud a una central de monitoreo, autos que reservan horas en el mecánico y un largo listado de ideas que hasta hace poco parecían sacadas de películas de ciencia ficción (Internet de las Cosas o IoT por sus siglas en inglés). Estas soluciones se han generado en todas partes, a distintas escalas, con un factor común: sus creadores sabían programar.

Debido a la utilidad concreta que ha demostrado el saber programar para solucionar los problemas de la vida cotidiana contemporánea, se ha producido un creciente interés por aprender a programar, enseñar a programar y contratar personas que sepan programar en lenguajes como Python, Java, Ruby o R, entre otros. Por ejemplo, de acuerdo al sitio PayScale (www.payscale.com) que compara las remuneraciones entre distintos oficios y profesiones en los Estados Unidos, un desarrollador de software y programador en Python puede aspirar a una remuneración anual sobre los cien mil dólares (esto es, sobre 6 millones de pesos mensuales). Diversas organizaciones privadas han entendido la necesidad de promover la enseñanza y el aprendizaje de programación, existiendo en la actualidad un amplio set de opciones pagadas y gratuitas para aprender a programar vía internet tales como studio.code.org, CodeAcademy.com, CodeSchool.com, Coursera.com, StackAcademy.com, GeneralAssemb.ly, Udacity.com, GirlsWhoCode.com y KhanAcademy.com por mencionar sólo algunas. Lo mismo ha sucedido con organizaciones públicas que han promocionado tanto el aprendizaje como el uso de las habilidades de programación para extraer valor a datos de interés público con proyectos tales como Codeforamerica.org en los Estados Unidos o Uk.code.org en el Reino Unido.

Algunos países han dado el paso de enseñar ciencias computacionales en las escuelas. Por ejemplo, el 2014, Inglaterra se convirtió en el primer país del grupo G7 en hacer obligatoria la enseñanza de ciencias computacionales en el currículum escolar para todos los estudiantes entre 5 y 16 años. La iniciativa Computing in the Core (www.computinginthecore.org) busca el mismo objetivo en los Estados Unidos en la misma dirección avanzan otras naciones como Dinamarca y Alemania. Por su parte, Estonia ha introducido la programación desde primero básico con iniciativas como las impulsadas por la Tiger Leap Foundation (véase http://koolielu.ee/).

La iniciativa Computing in the Core (www.computinginthecore.org) busca el mismo objetivo en los Estados Unidos y en la misma dirección avanzan otras naciones como Dinamarca y Alemania. Por su parte, Estonia ha introducido la programación desde primero básico con iniciativas como las impulsadas por la Tiger Leap Foundation (véase http://koolielu.ee/). Hacia comienzos del año académico del 2014, alguna forma de programación también era parte del currículum en otros países europeos como Bulgaria, Chipre, República Checa, Grecia, Irlanda, Italia, Lituania, Polonia y Portugal. Iniciativas similares existen en Australia (por ejemplo, http://www.codeclubau.org/ y http://www.codeforaustralia.org/) y Canadá (por ejemplo, http://ladieslearningcode.com/). La organización Code.org comenzó a divulgar la importancia de aprender a programar en el año 2013. A comienzos del 2016 implementó studio.code.org, una plataforma para que docentes enseñen a programar basado en el currículo. Dentro de las acciones masivas está la campaña mundial "La Hora del Código" una invitación que consiste en escribir las primeras líneas de programación y verificar que poder programar es posible. Code.org provee una plataforma donde todo el que quiera puede ensayar, escribir y realizar instrucciones a conocidos personajes de películas y juegos.

En Chile también se ha ido generando un movimiento en torno a la importancia de la programación, así desde 2011 existe una medición del estado de la alfabetización digital mediante el SIMCE TIC, y entre 2014 y 2015, se aprobaron los Proyectos Resolución N°47, N°392 y N°487 orientados a potenciar activamente el rol de la Programación y el Pensamiento Computacional (PyPC)[3] en el currículum educacional, y la innovación basada en tecnologías digitales.

Por otra parte, existen diversas iniciativas impulsadas por CORFO sobre innovación: el programa StartUp Chile, talleres extracurriculares en el programa Enlaces del Ministerio de Educación (MINEDUC) y programas como "Yo elijo mi PC", ambos focalizados en reducir la brecha digital; a su vez el programa "Tablet para la educación inicial", también de Enlaces, introduce la lógica matemática tempranamente, y al ser complementado con el robot "Albert", permite a párvulos avanzar en nociones de programación con tarjetas y material físico. También existe el proyecto "Mi taller Digital", el cuál integra la programación como taller optativo en sus modalidades de robótica y videojuegos. Todos estos programas se dan en el marco de las llamadas Habilidades TIC Para el Aprendizaje (HTPA), "que están orientadas a la capacidad de transformar la información o adaptarla para hacer un nuevo producto o desarrollar una nueva idea. Esta matriz ya está vinculada al currículum chileno (…) poniendo a los estudiantes como productores de conocimientos (más que como meros consumidores)"[4].

Por su parte BiblioRedes -dependiente de la Dirección Nacional de Bibliotecas, Archivos y Museos- lanzó el 2015 el programa "Taller de Jóvenes Programadores", una iniciativa que cuenta ya con más de 10.000 graduados en lenguajes de programación Scratch, JavaScript y App Inventor en distintos niveles, y en 2016 se incorporaron CSS y PHP.

[3] Iniciativas promovidas también desde el mundo académico y universitario, como la del Dr. Jorge Pérez, profesor asociado del Departamento de Ciencias de la Computación de la Universidad de Chile.

[4] Como así nos lo comentó José Gorrini, encargado de Enlaces del Departamento de Educación General del MINEDUC.

También existen iniciativas privadas, como las llevadas a cabo por el liceo tecnológico Enrique Kirberg de Maipú para enseñar a programar mediante el lenguaje Scratch; la Olimpiada Chilena de Informática; la "Plataforma Programa tus Ideas" de Samsung y País Digital; los TecnoTalleres propiciados por una alianza entre la Fundación Mustakis, INACAP e IBM; la "Hora del Código" en Chile, que ya contase con 22 mil participantes en el 2015, liderada por la fundación Kodea[5], la que por lo demás trabaja actualmente en un programa piloto de reconversión laboral femenino a través de la enseñanza de código en conjunto con Duoc y Corfo.

Entre otras iniciativas privadas, destacan las de Fundación Telefónica, como la "Comunidad de Programadores Robóticos" y "Comunidad Jóvenes Emprendedores" que reúne jóvenes con miras a propiciar el intercambio de experiencias provechosas para ellos y su formación y desarrollo. Así cuenta también con espacios como "Aulas Fundación Telefónica", el primero con el objetivo de contribuir a que niños y niñas desarrollen competencias para desenvolverse en una sociedad digital, y "Aula Lab", que ofrece un abanico de talleres abiertos a la comunidad educativa. Recientemente han lanzado la plataforma abierta STEM (Science, Technology, Engineering, Mathematics) para jóvenes entre 14 y 17 años. Se trata de una plataforma con cursos en ciencias y programación, la Fundación tiene como objetivo contribuir a instalar la programación en el currículo chileno.

Aunque las distintas iniciativas no necesariamente están orientadas a formar programadores, hay expectativas en la educación superior. Así, la necesidad de aprender a programar en la educación primaria y secundaria es vista como una necesidad por parte del mundo universitario. Por ejemplo, en la Universidad Nacional de Córdoba, Argentina, se preguntaban por qué los estudiantes de primer año de la carrera de ingeniería fracasaban en las pruebas parciales y finales produciendo una alta tasa de deserción. Bartó y Díaz[6], sostienen que existe consenso en identificar a la naturaleza formal de los lenguajes de programación como la mayor dificultad que tienen los estudiantes en aprender a programar. En efecto, la distancia que existiría entre los modelos mentales y modelos conceptuales, que se verifica en la incapacidad de abstraer y manejar con habilidad los lenguajes formales, se presentan como una dificultad insalvable en la enseñanza de la programación. Una de las razones, indican las investigaciones, se deriva del hecho que los estudiantes se encuentran con un déficit de educación lógico-formal que es considerada un riesgo para el éxito del desempeño en la informática. Una conclusión propone entrenar durante la enseñanza secundaria en procesos de argumentación y pensamiento crítico.[7] Por otra parte, en Inglaterra han buscado una explicación a la tasa de reprobación que tienen consistentemente los estudiantes en los cursos de enseñanza de programación a nivel universitario. Dehnadi y Bornat reportan que, si bien siempre se ha sospechado que hay personas que tienen aptitud para programar, ésta no está asociada ni a edad, género, nivel educacional ni tampoco a resultados de test de inteligencia o de resolución de problemas. Ellos ofrecen un test de aptitudes para la programación que entrega resultados robustos incluso "antes que los estudiantes hayan tenido contacto con algún lenguaje de programación"[8]; apoyando la idea de que en la educación secundaria debería iniciarse la enseñanza de pensamiento computacional.

En suma, saber programar se ha convertido en una tendencia, como se ha planteado en los párrafos precedentes, existen buenas razones para creer que ello haya finalmente sucedido cerca de medio siglo después de que los primeros impulsores de la programación levantaran la voz a mediados de la década de 1960.

---

[5] La Hora del Código es organizada el año 2016 por Kodea, Ucorp y el Ministerio de Economía, cuenta con financiamiento parcial de Corfo.

[6] Carlos Alberto Bartó, Laura Cecilia Díaz. Proyecto: Sistemas Inteligentes Aplicados a las Enseñanza de la programación en la Ingeniería. XIV Workshop de investigadores en ciencias de la computación.

[7] Carlos Alberto Bartó, Laura Cecilia Díaz. El déficit en la formación lógico-formal como factor de riesgo en el desempeño en Informática. Latin American and Caribbean Journal of Engineering Education, Vol. 2(1), 2008.

[8] Saeed Dehnadi and Richard Bornat. The camel has two humps (working title) School of Computing Middlesex University, UK 2006.

2.- Origen de la idea: Aprender a programar en la escuela.

Los conocimientos previos, que se adquieren en la escuela, deben servir para resolver problemas nuevos. Así, enseñar a los estudiantes a transferir conocimiento es un objetivo que siempre ha estado en la educación escolar. Hasta avanzado el siglo XX se proponía la enseñanza del latín como instrumento de promoción de disciplina mental y de pensamiento ordenado. A mediados del siglo pasado, la enseñanza de la matemática cumplió ese rol.[9]

A comienzos de la década de 1960, algunos visionarios empezaron intensas campañas en las que proponían que el aprendizaje de la programación estuviese al centro de una revolución educacional y no tan sólo de una reforma de la enseñanza escolar. Lo que esos pioneros vieron fue el potencial de la enseñanza de la programación para enfrentar una realidad nueva que se estaba instalando rápidamente en las sociedades contemporáneas: mientras afuera de la escuela, los estudiantes estarían expuestos a conocimientos y estímulos por múltiples medios (texto, imagen, video, sonido), con contenidos locales y globales, instantáneos y simultáneos, las rutinas escolares tradicionales los seguirían impulsando hacia una forma de enseñanza-aprendizaje fundamentalmente lineal, con un énfasis en la presencia de un profesor como fuente central de transmisión de saber. Tal disonancia permitía anticipar que las prácticas tradicionales de enseñanza se convertirían en desmotivadoras y poco desafiantes para las nuevas generaciones. En ese contexto, cada vez se tornaba más clara la necesidad de cambiar el foco del proceso educativo desde la transmisión de conocimientos hacia la estimulación del proceso de enseñanza-aprendizaje en los estudiantes, de forma tal que estos se convirtieran en sujetos activos de su aprendizaje, y los profesores en facilitadores del mismo. Dos marcos teóricos permitieron dar sustento conceptual a esta visión constructivista del aprendizaje. Por un lado, se puso atención en los modos en que los estudiantes van asimilando y adquiriendo conocimiento desde sus propios modos de interpretar y explorar el mundo (por ejemplo, mediante la influencia de los estudios de Jean Piaget) y, por el otro, se dio relevancia a los procesos socioculturales que enmarcan el desarrollo cognitivo (por ejemplo, mediante el descubrimiento y apropiación en occidente de la obra de Lev Vigotsky). Sobre la base de estos marcos teóricos, se empezó a construir un currículum en el que el proceso educativo pudiese estimular a niños y niñas con acciones que tuviesen un sentido para ellos. Desde la década de los noventa, los planes y programas oficiales empezaron a hacer cada vez mayor hincapié en que los estudiantes son quienes construyen activamente su conocimiento, pero que esa construcción no la hacen solos sino en el contexto donde se desenvuelven, con sus especificidades culturales y socioeconómicas.

A comienzos de la década de 1960, algunos visionarios empezaron intensas campañas en las que proponían que el aprendizaje de la programación estuviese al centro de una revolución educacional y no tan sólo de una reforma de la enseñanza escolar.

[9] Meyer, R. Learning and Instruction, Pretince Hall. 2003.

Papert desarrolló el lenguaje Logo para crear un espacio que permitiera justamente lo contrario: que los estudiantes sean los que programen los computadores. En sus palabras, al aprender a programar, el estudiante "piensa sobre cómo pensar, hace que el niño se convierta en un verdadero epistemólogo".

Influido por estas concepciones constructivistas del aprendizaje y siendo un experto en inteligencia artificial, Seymour Papert visualizó a comienzos de los 1960s el potencial que tenía el lenguaje computacional para inducir y catapultar esos desarrollos cognitivos y se abocó a promover el uso de la computación y la programación en el aprendizaje. En Boston, Estado Unidos, tres expertos en informática se reunieron para desarrollar un lenguaje computacional que sirviera para enseñar a los escolares a programar[10]. Las promesas era dos: i) Los alumnos de tercero básico serían capaces de hacer tareas simples con muy poca preparación; ii) La estructura de programación tendría incorporados conceptos matemáticos.

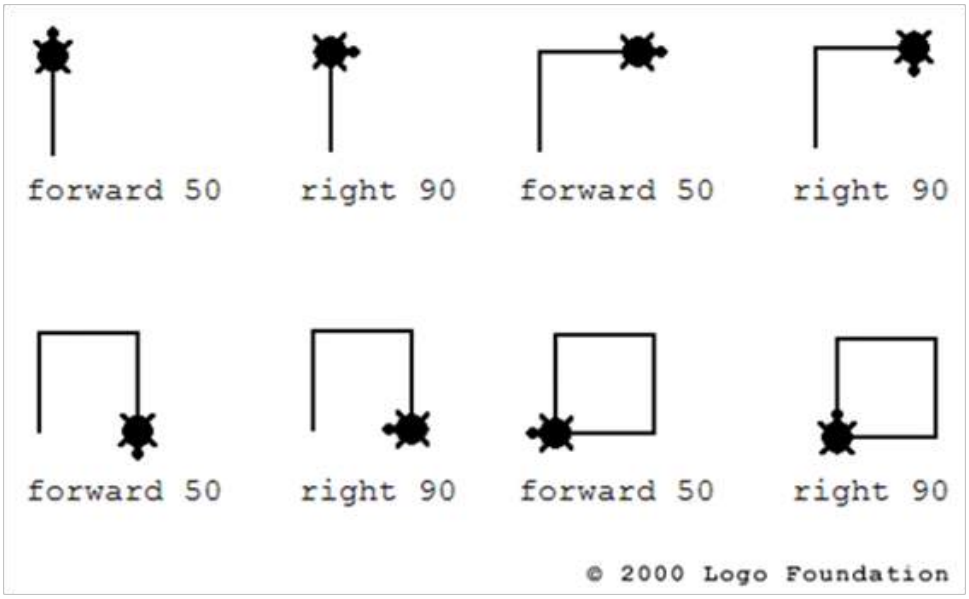
Desde entonces, Papert ha sido un ferviente impulsor de la enseñanza de la programación como una forma de expandir el aprendizaje en el estudiantado. Luego de dos décadas de intensa promoción de la enseñanza de la programación, a comienzos de la década de los noventa del siglo pasado, Papert escribía en su obra Mindstorms, Computers and Computer Cultures, que el uso de los computadores en las escuelas había sido básicamente para proveer de ejercicios, retroalimentar y entregar información más que herramientas para que los estudiantes expandieran sus horizontes de aprendizaje. Para Papert, debido a la forma en que se usaban los computadores en las escuelas, eran los computadores los que programaban a los estudiantes y no al revés. Fuertemente influenciado por la mirada constructivista de Piaget (trabajó con él), Papert desarrolló el lenguaje Logo para crear un espacio que permitiera justamente lo contrario: que los estudiantes sean los que programen los computadores. En sus palabras, al aprender a programar, el estudiante "piensa sobre cómo pensar, hace que el niño se convierta en un verdadero epistemólogo". El estudiante se transforma en un constructor que construye a partir de sus propios materiales, principalmente a partir de las metáforas y modelos conceptuales que le sugiere su entorno cultural.

Para Papert, la programación permite enseñar a los niños y niñas a aprender sin ser enseñados, es decir, a aprender a aprender. Aquí hay dos ideas claves en la propuesta de Papert: i) programar permite un aprendizaje auto guiado; y ii) programar compromete al estudiante a aprender acerca de su propio aprendizaje. Por este motivo, Papert considera que los computadores representan una revolución tecnológica superior a otras que existieron durante el siglo XX como la televisión. En los tiempos en que aún no existían computadores personales, Papert ya visualizaba que tarde o temprano ellos serían masivos y que podían ser utilizados para crear condiciones donde los menores podían adquirir pensamiento formal mediante juegos que de otro modo serían muy abstractos para su nivel de desarrollo. Mediante Logo, Papert intentó traducir lo abstracto (conceptos lógicos, conceptos matemáticos, etc.) en acciones concretas que le permitiría a los estudiantes tomar conciencia de sus propios modos de aprendizaje mediante la experimentación y el juego. Aprendiendo cómo pensar más que aprendiendo qué pensar, por ejemplo, mediante el ejercicio de programar jugando, los niños son capaces de descubrir las reglas de la aritmética o de las fracciones.

[10] Logo fue creado por Papert, Feurzeing y Bobrow en 1966. La investigación acerca de Logo comenzó en BNN Technologies y luego de 4 años continuó en el MIT. Logo a Project History, MIT.

La figura 1 presenta un ejemplo de programación en Logo. Un cursor en forma de tortuga se desplaza (por ejemplo, forward 50, forward 10, etc.) o gira (por ejemplo, right 90, left 15) de acuerdo a las instrucciones dadas por el usuario. De forma tal que dando las instrucciones que aparecen en la figura es posible construir, por ejemplo, un cuadrado.

Figura 1: Ejemplo del uso del lenguaje de programación Logo



Fuente: Logo Foundation

En el entorno creado por Logo, las ideas nuevas son habitualmente generadas por el propio usuario a medida que busca formas de satisfacer necesidades particulares. Por ejemplo, las variables matemáticas son usualmente enseñadas en las escuelas en el contexto de una ecuación (por ejemplo: a*b = 100). Papert sostiene que una formulación como esa del concepto de variable carece de contexto de interés para el estudiante, lo desmotiva y puede incluso inducir rechazo hacia la matemática. En cambio, en un entorno como Logo, si el estudiante desea crear, por ejemplo, un rectángulo de algún área específica y ancho predefinido, usará el concepto de variable aun cuando no lo comprenda plenamente mediante su ejercicio de crear el rectángulo. Es en este contexto, señala Papert, que el concepto pasa de ser una abstracción a una herramienta para resolver un problema concreto.

Siguiendo a Piaget, Papert sostiene que el aprendizaje de los números, la aritmética o las fracciones y en general del conocimiento, es progresivo en el sentido que las personas adquieren nociones que van mejorando a medida que van interactuando y haciendo sentido de los conceptos hasta que la idea es finalmente descubierta por ellos. Si bien Piaget no tuvo éxito en sus aseveraciones sobre qué comportamientos específicos son innatos, Papert sostiene que sí lo fue en comprender el carácter evolutivo del aprendizaje y que todo análisis epistemológico debe dar cuenta de dicha evolución desde los primeros aprendizajes hasta que las ideas se plasman en las personas. Pese a su entusiasmo y dedicación, Papert no tuvo el éxito al que aspiraba. En parte, ello se debió a que la necesidad y la ventana que abre la programación seguían siendo latentes para muchos; de modo tal que decir que programar es una habilidad básica no parecía convencer a muchos en las décadas de 1970 y 1980. Pero las dificultades para convencer que enfrentó Papert también se debían en parte al propio lenguaje Logo.

Siguiendo a Piaget, Papert sostiene que el aprendizaje de los números, la aritmética o las fracciones y en general del conocimiento, es progresivo en el sentido que las personas adquieren nociones que van mejorando a medida que van interactuando y haciendo sentido de los conceptos hasta que la idea es finalmente descubierta por ellos. Si bien Piaget no tuvo éxito en sus aseveraciones sobre qué comportamientos específicos son innatos, Papert sostiene que sí lo fue en comprender el carácter evolutivo del aprendizaje y que todo análisis epistemológico debe dar cuenta de dicha evolución desde los primeros aprendizajes hasta que las ideas se plasman en las personas. Pese a su entusiasmo y dedicación, Papert no tuvo el éxito al que aspiraba. En parte, ello se debió a que la necesidad y la ventana que abre la programación seguían siendo latentes para muchos; de modo tal que decir que programar es una habilidad básica no parecía convencer a muchos en las décadas de 1970 y 1980. Pero las dificultades para convencer que enfrentó Papert también se debían en parte al propio lenguaje Logo.

Es por esto que esfuerzos posteriores han intentado generar otros lenguajes y métodos para enseñar a programar a niños y niñas. En el 2003, Caitlin Kelleher y Randy Pausch de Carnegie Mellon University, elaboraron una taxonomía de lenguajes de programación disponibles a la fecha distinguiendo dos propósitos de enseñanza de cada lenguaje: aprender a programar como un fin en sí mismo o aprender a programar como un objetivo intermedio en la búsqueda de otro fin.[11] El segundo grupo, que es el de interés en este documento, lo denominaron "sistemas empoderadores" y se presenta en la Tabla 1. Los autores distinguen dos finalidades en la creación de lenguajes para otros fines, que el de aprender a programar como fin en sí mismo: (i) usar los mecanismos de programación para que se impulsen desarrollos cognitivos y (ii) realizar actividades mediante la programación. Como se puede ver en la tabla, Logo puede clasificarse como un lenguaje que busca fomentar el desarrollo cognitivo desde la premisa que la principal dificultad de la programación radica en el lenguaje o sintaxis utilizada y, por ende, pretende hacer del lenguaje más comprensible.

[11] Kelleher, C y R. Pausch (2003): "Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers" ACM Computing Survey 37,2, pp 83-137.

Tabla 1: Taxonomía de lenguaje de programación como empoderadores para fines distintos de la programación propiamente tal.

Fines de su creación	Hipótesis de trabajo	Método	Lenguaje
Mecanismos de programación para mejorar el desarrollo cognitivo	Vencer las barreras que impone la dificultad del código	Acciones demostrativas en la interface	Pygmalion, Rehearsal, Mondrian
		Condiciones y acciones demostrativas	AgentSheets ChemTrains Stagecast
		Acciones específicas	Pinball Construction Set Alternate reality kit Klik N Play
	Mejorar el lenguaje de programación	Hacer del lenguaje más comprensible	COBOL Logo Alice 98 HANDS
		Mejorar la interacción con el lenguaje	Body Electric Fabrik Tangible Programming con Trains Squeak cToys Alice 99 AutoHAN Physical Programming Flogo
		Integración con el entorno	Boxer Hypercard cT Chart N Art
Actividades resaltadas mediante la programación	Entretenimiento	Bengo Mindrover	
	Educación	SOLO Gravitas Starlogo Hank	

Fuente: Adaptado de Kelleher y Pausch (2003)

En esa categoría, Logo comparte con COBOL (diseñado para facilitar la creación de aplicaciones para negocios), Alice98 (creado para generar gráficos en tres dimensiones accesible para estudiantes universitarios) y HANDS (creado para que los niños desde quinto grado puedan hacer juegos). No obstante, la tabla 1 también refleja otras posibles hipótesis de partida desde las cuales se han propuesto lenguajes. Por ejemplo, bajo la premisa que lo que se debe buscar es superar las barreras de la codificación, Pygmalion (1975) fue el primer lenguaje basado en demostraciones. Los usuarios creaban programas implícitamente mientras trabajaban en un ejemplo (típicamente, de operaciones matemáticas). Otros programas permiten a los usuarios crear condiciones para que se ejecuten las acciones. Tal es el caso de, por ejemplo, Stagecast (1995) que permite hacer simulaciones a partir de las reglas definidas por el usuario. Un tercer grupo de lenguajes están orientados a la ejecución de acciones ordenadas por el usuario sin que este haya tenido que escribir un código. En ese conjunto se encuentra el programa Klik N Play que entrega la posibilidad a los usuarios de crear sus propios objetos (y animaciones de esos objetos cuadro por cuadro) mediante la selección de botones y definiciones de acciones aplicables a esos botones.

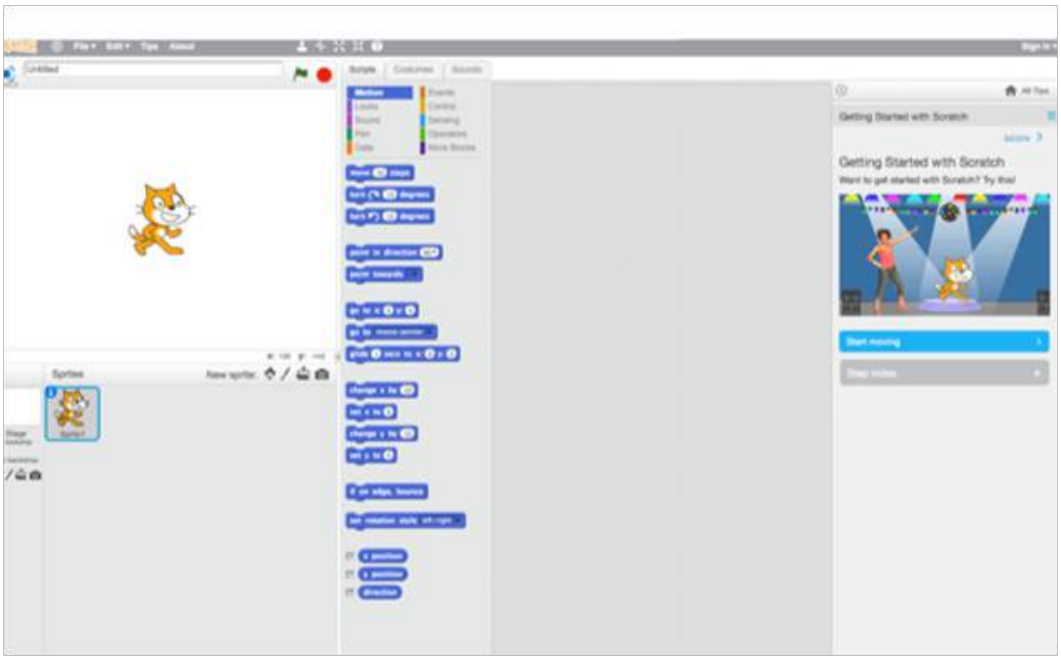
Entre los lenguajes que fueron creados para que los usuarios aprendan a escribir programas con otros fines que la propia programación también existen casos en que se facilita la interacción con el lenguaje, usando paquetes prefabricados de instrucciones que el usuario puede seleccionar. Tal es el caso de Fabrik (1988). Fabrik es un set de procedimientos en cajas que pueden ser organizadas en distintas formas por el usuario para crear programas.

Otros programas intentan facilitar aún más el proceso de programación integrando la misma a entornos computacionales donde la persona es usuaria. Por ejemplo, Chart N Art (1996) es un programa que se podía utilizar como complemento de MacDraw para mejorar la visualización de gráficos.

En la década del 2000, nuevos programas fueron creados bajo criterios que bien pueden ser incorporados en las taxonomías anteriores. Entre ellos, el esfuerzo que quizás más éxito relativo ha obtenido y que debería ubicarse en la misma casilla que Logo en la tabla anterior es el impulsado por Mitch Resnick desde el MIT, denominado Scratch (2003)[12]. Inspirado en el carácter modular del juego Lego, Scratch funciona en bloques de programación que se pueden montar unos sobre otros como piezas de Lego. La figura 2 presenta un ejemplo de una interfaz de Scratch. El área para programar emula un escritorio de trabajo donde puede haber elementos sin uso. Siguiendo la idea avanzada por Logo, el programa prioriza la personalización de su uso permitiendo alta flexibilidad en el trabajo con imágenes, sonido y creación de figuras. Adicionalmente, el programa tiene un fuerte componente social, pues está integrado con un sitio web que permite generar una comunidad en línea donde los usuarios pueden encontrar apoyo, colaboración, retroalimentación y ver mutuamente los programas que han hecho. De acuerdo a Resnick, se suben al sitio web diariamente más de 1500 nuevos proyectos con códigos libres para ser reutilizados y combinados, incluyendo videojuegos, cartas interactivas, simulaciones científicas, tours virtuales, tarjetas de cumpleaños, etc. con una audiencia principalmente entre los 8 y los 16 años de edad.[13]

[12] Resnick desarrolló Lego Logo a mediado de los 80´s, Actualmente es Lego Papert Professor of Learning Research en el MIT.
[13] Véase Resnick et al (2009): “Scratch: Programming for all” Communications of the ACM, Noviembre, Vol 52, No.11.

Figura 2: Ejemplo del uso del lenguaje de programación SCRATCH



Recientemente se han propuesto otros métodos para enseñar habilidades específicas a niños en edad preescolar y en los primeros años de escolaridad, como secuenciación mediante la programación de robots.[14] Fisher Price, compañía especializada en juguetes, lanzará este año su nuevo juguete Code-a-Pillar que promete hacer "Jugar y Pensar" a niños desde los 3 años, pues pueden hacer andar a su oruga programándolas fácilmente.

[14] Véase por ejemplo Kazakoff, Sullivan y Bers, "The Effect of classroom-based intensive robotics and programming workshop on sequencing ability in early childhood" en Early Childhood Education Journal (2013), vol 41, pp 245-255; Kazakoff y Bers, "Put your robot in, put your robot out: Sequencing through programming robots in early childhood" en Journal of Educational Computing Research, vol 50, No.4, pp. 553-573 y Sullivan y Bers, "Robotics in the early childhood classroom: learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade" en International Journal of Technology Design and Education, 2015.

El avance en la tecnología y la reducción de costos ha hecho posible el desarrollo de robots que responden a instrucciones de los niños mediante manipulación de bloques con instrucciones que pueden ubicarse secuencialmente. La principal innovación de estos proyectos es que trasladan la enseñanza de la programación desde la pantalla a la actividad de juego en la que los estudiantes se desarrollan naturalmente en sus primeros años de escolaridad.

En resumen, en aproximadamente cincuenta años de desarrollo de lenguajes de programación y con el objeto de facilitar el desarrollo cognitivo, los promotores de la enseñanza de la programación en las escuelas han considerado que aprender a programar tiene al menos los siguientes beneficios para los niños y niñas en edad escolar y preescolar:

- Crean su propio entorno de aprendizaje, aprendiendo paso a paso a comprender sus propios modos de aprender.
- Aprenden conceptos matemáticos mediante su uso previo al desarrollo o comprensión abstracta de los mismos.
- Reducen las reticencias hacia conceptos abstractos, particularmente de origen matemático.
- Aprenden a organizar y secuenciar tareas en forma lógica.
- Desarrollan una actitud positiva hacia el aprendizaje.
- Aprenden a colaborar con otros en la resolución de problemas.
- Se empoderan en el proceso de aprendizaje.

Ahora bien, ¿existe alguna evidencia en favor de todos o algunos de estos beneficios que promueven los partidarios de introducir la programación de las escuelas? En la próxima sección se revisan algunos resultados de investigaciones empíricas que se han desarrollado al respecto.

3.- Impactos del aprendizaje de programación en las escuelas.

Existe abundante literatura empírica sobre los eventuales impactos de aprender a programar en menores en edad escolar, pudiendo distinguirse tres grandes tipos de impactos esperados: (a) cognitivos, (b) de aprendizaje y (c) sociales. Los impactos cognitivos se refieren al desarrollo de habilidades para la resolución de problemas, habilidades matemáticas, conocimiento de conceptos matemáticos y de programación y habilidades para evaluar el propio proceso de aprendizaje. Entre los impactos en el proceso de aprendizaje se incluyen cambios en los niveles de motivación, involucramiento, predisposición positiva/negativa para el aprendizaje. Por último, los impactos sociales se refieren a la extensión y naturaleza de las interacciones con otros a medida que se aprende a programar.

8 razones por las que enseñar programación en colegios

Mejora la capacidad de entender conceptos y abstracciones de tipo matemáticas	(Gibson, 2012)
Desarrolla el pensamiento lógico secuencial y la resolución de problemas complejos	(Passey, 2016)
Mientras antes y más a menudo se introduzcan contenidos de programación, más aumenta la propensión a utilizarlos en la resolución de problemas	(Syslo & Kwiatkowska, 2015)
Las nuevas interfaces de programación en bloque como Scratch, facilitan mucho el aprendizaje y lo hacen divertido	(Resnick, et al, 2009)
Manejar programación te permite dialogar activamente y mantenerte actualizado con las transformaciones tecnológicas mundiales, pese a su gran rapidez	(Partelow, 2016)
Los nuevos trabajos que se están creando, se basarán esencialmente en tecnologías de la información, la programación es la herramienta para interactuar con estas	(WEF, 2016)
Se está utilizando cada vez más para resolver problemas reales de las personas, con resultados alentadores	(The Economist, 2016)
La complejidad de la toma de decisiones algorítmicas planteará nuevos desafíos para la sociedad civil futura; entender programación será fundamental para resguardarse de nuevas formas de discriminación ocultas en líneas de código	(Goodman & Flaxman, 2016)

Uno de los programas sobre los cuales más investigación se ha realizado en cuanto a sus impactos es Logo. Por ejemplo, entre los estudios en la primera mitad de la década de 1980 sobre el impacto de la programación destaca el desarrollado por Clements y Gullo (1984). Ellos analizan los efectos de aprender a programar con Logo en menores de 6 años de edad en relación a cuatro dimensiones: a) el desarrollo cognitivo, b) las habilidades meta cognitivas, c) el estilo cognitivo y d) la habilidad para describir direcciones. El estudio se basa en una muestra pequeña de 18 estudiantes de seis años de edad que fueron divididos en dos grupos con igual número de niños y niñas (a un grupo se les enseñó Logo, mientras que el otro trabajó en actividades asistidas por computadoras). Los autores encontraron evidencia a favor de la enseñanza de la programación en el desarrollo de mejores estilos cognitivos y en la habilidad para describir pasos lógicos (secuenciación).

Por su parte, Gloria Miller y Catherine Emihovich, ambas de la Universidad de South Carolina, realizaron experimentos con catorce menores en edad preescolar en once sesiones de programación en un transcurso de tres semanas de entrenamiento en Logo, encontrando incrementos en la habilidad de monitorear el propio aprendizaje (específicamente, identificar qué no entienden y el por qué no lo entienden) tras la participación en el programa. En otro estudio en el que Logo estaba incorporado como parte de las actividades, Hilda Carmichael y colaboradores estudiaron por dos años una muestra de 433 estudiantes, 13 profesores y 18 cursos encontrando que el uso creativo de los computadores había impulsado el desarrollo de pensamiento de forma independiente y había contribuido a generar un ambiente propicio para la exploración de ideas e interacción entre los estudiantes. Por otro lado, el estudio también encontró efectos en términos de disposición a expresar, refinar y revisar ideas y un aumento de la estima de los estudiantes. En otra investigación, Douglas Degelman y colaboradores asignaron a 15 preescolares en dos grupos, de forma tal que a un grupo se le enseñó LOGO y al otro no, obteniendo diferencias cognitivas en favor del primer grupo. Específicamente, los autores encuentran que la proporción de respuestas correctas en la solución de tareas era mayor en ese grupo.

No obstante, los resultados comentados es en el párrafo anterior, los impactos de los primeros estudios en la década de 1980 sobre el pensamiento lógico o la solución de problemas más allá de los relativos a la programación no fueron todos concluyentes. Por ejemplo, Michael Battista y Douglas Clements asignaron estudiantes aleatoriamente a tres grupos (uno donde se enseñaba Logo, otro donde se hacían actividades asistidas por computadores y un tercer grupo donde las mismas actividades se realizaban de la forma tradicional), el estudio no encontró diferencias significativas en el desempeño matemático ni el desarrollo de habilidades relativas a la independencia en el aprendizaje, aunque sí hubo diferencias en beneficio del grupo que aprendió Logo en lo relativo a la solución de problemas. El estudio de Douglas Degelman comentado en el párrafo anterior, si bien encontró resultados positivos, el análisis estadístico de varios de sus datos mostraba que sus conclusiones no tenían el suficiente poder explicativo como para ser estadísticamente significativas.

Estos resultados sugirieron escepticismo y alimentaron la impresión que existía un efecto no significativo de la programación. Un Meta análisis de estos estudios sugiere varias explicaciones para la ausencia de impacto significativo. Primero, la mayoría de las investigaciones se concentran en los aspectos formales de los conceptos de programación, de modo tal que al indagar sobre la penetración de esos conceptos en otros ámbitos de la vida de los estudiantes, no se observaron incidencias. Pero ello podría deberse precisamente a que no es en su formalidad conceptual que inciden en el aprendizaje. O bien, la falta de efectos podría atribuirse no necesariamente a una ausencia de impacto de la programación, sino a que es aprendida fuera de los contextos habituales de los estudiantes. Segundo, la extensión de la exposición a situaciones de programación fue en lo general corta; por ende, la ausencia de impactos podría estar asociada precisamente a que no se alcanzó un mínimo requerido para que generara algún efecto.

En tercer lugar, no sólo la exposición a situaciones de programación podrían haber sido insuficientes, sino también los programas desarrollados por los estudiantes fueron en general de corta extensión. Nuevamente, ello podría implicar que no se alcancen impactos suficientes para superar criterios estadísticos estándar.

En la segunda mitad de la década de 1980, los estudios acotaron mejor los focos de la investigación y los propósitos en el uso del programa Logo. En particular, se empezó a utilizar el programa para enseñar habilidades especialmente dirigidas a resolver problemas. Por ejemplo, en 1987, Sharon Carver[15]desarrolló un estudio en el que enseñaba cómo solucionar un problema de los comandos de programación en Logo y luego testeó con éxito que los estudiantes eran capaces de utilizar lo aprendido para resolver problemas fuera del entorno de Logo. Otras intervenciones posteriores con Logo hicieron hincapié en exponer a los estudiantes a destinar tiempo considerable de trabajo en tareas de programación y de resolución de problemas con similares impactos en la apropiación de habilidades. Un proyecto de especial interés fue el desarrollado por Idit Harel en 1988[16], como parte de su trabajo de tesis doctoral, la investigadora creó el Instructional Software Design Project (ISDP) para que estudiantes de cursos superiores enseñaran fracciones a sus compañeros de cursos inferiores mediante el uso del programa Logo. El utilizar el programa para enseñar fracciones involucra más que la mera producción de código y, por ende, se acerca más a la experiencia de la generación de un producto y, en último término, implica la reformulación de procesos cognitivos de un contexto a otro. Los resultados fueron positivos. Los estudiantes no sólo aprendieron Logo, también fracciones, enseñar fracciones y diseño de software.

Debido a que muchos de estos estudios están basados en muestras pequeñas, a comienzos de los años 1990 todavía persistía el escepticismo respecto de los impactos de la programación en el aprendizaje. En ese contexto, los investigadores Yuen-KuangLiao y George Bright de las universidades de Houston y North Carolina (Greensboro)[17], desarrollaron un meta análisis de sesenta y cinco estudios experimentales en la década anterior. Para llevar adelante su trabajo, Liao y Bright estimaron la magnitud relativa de los efectos de cada estudio, definido dicho efecto como la diferencia de la media entre el grupo de tratamiento (que fue intervenido con la enseñanza de programación) y el grupo de control (que no recibió la enseñanza de programación) dividido por la desviación estándar del grupo de control. De acuerdo a sus resultados, en cincuenta y ocho estudios (89% de los casos) hubo efectos positivos estadísticamente significativos en favor de la enseñanza de la programación.

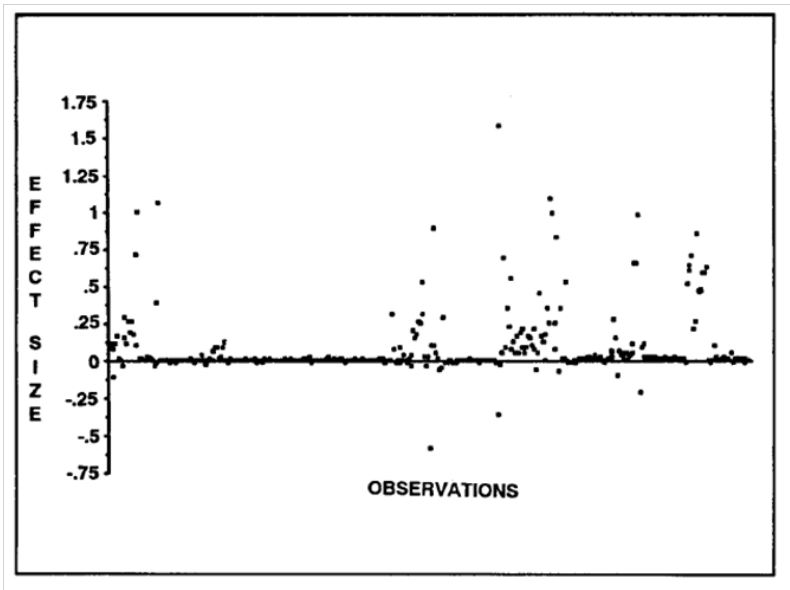
15 Véase Carver, S. “Transfer of Logo debugging skill: analysis, instruction, and assessment”, Computer Systems Group Bulletin, 14,(1), 4-6. 1987.

16 Véase Harel, I. “Children Designers: Interdisciplinary Constructions for Learning and Knowing Mathematics in a Computer-Rich School (Cognition and Computing Series)”, Norwood: Ablex.

17 Véase Liao, y G. Bright: “Effects of computer programming on cognitive outcomes: a meta analysis”. Journal of Educational Computing Research, Vol 73, p. 251-268, 1991.

Al contrario de lo sugerido en párrafos anteriores, los mayores impactos se obtuvieron en estudios de corta duración, en lugar de los estudios donde los estudiantes fueron sometidos a semestres completos de cursos de programación. En particular, se obtuvieron menos efectos globales en estudiantes de enseñanza media que en otras etapas del ciclo escolar. Ahora bien, estas diferencias podrían explicarse porque a nivel de enseñanza media, los efectos marginales son menores debido a la mayor exposición que los estudiantes tienen a computadores a esa edad respecto de los estudiantes en cursos inferiores. Por último, respecto de los lenguajes utilizados, los resultados de Liao y Bright muestran mayores efectos de Logo que de otros programas (por ejemplo: Basic o Pascal.)

Figura 4: Tamaño comparado de los efectos de la programación en resultados cognitivos en una muestra de 65 estudios



Fuente: Liao y Bright (1991), página 260.

Si bien a comienzos de la década de 1990, la evidencia en favor de la enseñanza de la computación como modo de potenciar el desarrollo cognitivo comenzaba a ser robusta, las prácticas escolares profundizaron el uso de los computadores como herramientas para la transmisión de conocimiento, acceso a información y asistencia en el proceso de enseñanza. Las dificultades ya referidas en las sintaxis de lenguajes como Logo, inclinaron a las escuelas, los profesores y los estudiantes a ser usuarios de las computadoras, más que programadores de las mismas. Los programas se tornaron cada vez más amigables para los usuarios y los nuevos softwares permitieron expandir ampliamente el uso de los computadores sin necesidad de aprender a programar. Por eso, los desarrollos continuaron, pero la enseñanza de lenguajes de programación no logró la penetración que sus promotores esperaban y se mantuvo como una actividad especializada. Como se indicó en la sección uno, no será sino hasta la exposición masiva y globalizada de datos y la consiguiente necesidad de dar sentido a ellos, que la demanda latente por aprender a programar se masificaría, alcanzando el momentum en que se encuentra actualmente.

En el intertanto, programas como Scratch han mostrado resultados positivos en términos de aporte al desarrollo cognitivo de los estudiantes. Por ejemplo, Kyungwon Koh de la Universidad de Oklahoma identifica cualitativamente tres cambios conductuales positivos derivados del uso de Scratch en estudiantes de nivel primario. Primero, en lo relativo a la creación de información en formatos digitales; segundo, en la búsqueda de visualizaciones apropiadas para facilitar la comunicación de contenidos a otros usuarios; tercero, en el desarrollo de la capacidad de combinar, organizar y reutilizar significativamente contenidos de forma creativa y; por último, en el desarrollo de una perspectiva modular del aprendizaje (esto es, la capacidad de construir contenidos en bloques autocontenidos que se pueden agrupar y desagrupar creativamente).

Todo ello, lleva al autor a concluir que Scratch logra empoderar a los estudiantes en el sentido deseado por Seymour Papert. Por su parte, David Malan y Henry Leitner llegan a resultados similares al introducir la enseñanza de Scratch en escuelas de verano para preuniversitarios como fase previa a la enseñanza de Java en Harvard University.[18] De acuerdo a los autores, sólo los estudiantes que ya tenían previa exposición a lenguajes de programación no vieron utilidad en la exposición previa a Scratch. En Preparing the Next Generation of Computational Thinkers, el creador de Scratch, Mitch Resnick y sus colaboradores -basados en los buenos resultados que se han obtenido en el aprendizaje de este lenguaje- proponen ir un paso más allá y potenciar su capacidad de generar espacios colaborativos a gran escala, aprovechando el fuerte componente de redes comunitarias que ha rodeado desde un principio al desarrollo del programa.

Así, en la escuela primaria y secundaria de Estados Unidos y otros países, se ha dado más interés en complementar el currículo con los beneficios cognitivos de Scratch. De manera alineada con el currículo, un estudio exploratorio realizado por Burke y Kafi[19]en la ciudad de Filadelfia implementó y evaluó la incorporación de Scratch en la enseñanza de escritura en una clase de enseñanza básica (7to-8vo básico) a lo largo de once talleres de escritura durante siete semanas de una clase. Cada taller comenzó con una mini lección que vinculó un procedimiento de programación con un estándar académico. Para los diez estudiantes participantes, el análisis cualitativo de los artefactos de escritura y programación, tanto como encuestas, entrevistas, y observaciones de clase, muestra que aprendieron los fundamentos de programación y los elementos de narración en paralelo junto con la terminología análoga. Como el uso de Scratch fue conducido por las convenciones de narración enmarcadas en los estándares y talleres de escritura, la desventaja fue el relativo limitado aprovechamiento del lenguaje de programación, ya que algunos conceptos matemáticos (condiciones, Boolean logic, variables) no coincidieron con la lógica de las narraciones lineales producidas. Sin embargo, la intervención resultó en evidencia a favor de la conexión entre la programación y el escribir como procesos interrelacionados de la composición. Hoy día el mundo Scratch incluye sitios llamado Scratch-Ed (http://scratched.media.mit.edu) para educadores, buscando incorporar la programación en su práctica pedagógica.

[18]Malan, D. y H. Henry: “Scratch for budding computer scientists”. SIGCSE´07, 2007, Covington, Kentucky, USA.

[19]Burke, Q., & Kafai, Y. B. (2012). The writers’ workshop for youth programmers. In Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE ’12 (p. 433). New York, New York, USA: ACM Press.

4.- Conclusiones y desafíos para la enseñanza de la programación

¿Por qué hay que prestar atención a la enseñanza de la programación en las escuelas? En este documento hemos presentado algunas tendencias, argumentos y evidencias con el objetivo de estimular el debate e interés en estas materias. En efecto la visión de Papert y otros, hace ya 50 años, era correcta: aprender a programar tiene impactos en el aprendizaje. La evidencia sugiere que mediante la programación las y los estudiantes aprenden a crear su propio entorno de aprendizaje, comprendiendo sus propios modos de aprender; conceptos matemáticos, mediante su uso previo al desarrollo o comprensión abstracta de los mismos; a organizar y secuenciar tareas en forma lógica; a desarrollar una actitud positiva hacia el aprendizaje; a colaborar con otros en la resolución de problemas; a empoderarse del proceso de aprendizaje.

No obstante, en los tiempos en que ellos plantearon esas ideas, faltaba la centralidad en el uso de computadores para el quehacer diario de las personas. Esa falta de centralidad contribuyó a que fuera más eficaz aprender a ser usuario que aprender a programar. Adicionalmente, no existía el acceso prácticamente ilimitado a información que permitiría masificar y globalizar el desarrollo de soluciones basadas en programación. Hoy, en cambio, esas condiciones existen. Muchas actividades diarias no pueden realizarse si no es gracias a computadores y la comunicación y flujo de información diaria es inconmensurable.

En dicho contexto, aprender a programar ya deja de ser una habilidad adicional a la que se puede o no tener acceso y comienza a ser una habilidad básica que hace la diferencia entre acceder o no al conocimiento y las oportunidades en forma pertinente. En este mundo más complejo e interconectado ya no es posible recurrir a soluciones estandarizadas, la necesidad de nuevas formas de solucionar problemas de la vida en sociedad crece al mismo tiempo que crece la información, la interconexión y la centralidad de los computadores en nuestras vidas.

Telefónica

FUNDACIÓN

www.fundaciontelefonica.cl