Actas de las Jenui, vol. 8. 2023.
Páginas: 269-276

Mejora de una asignatura para la formación del profesorado en programación basada en bloques

J. Ángel Velázquez Iturbide, Maximiliano Paredes Velasco,
Sergio Cavero Díaz, Daniel Palacios-Alonso
Departamento de Informática y Estadística
Universidad Rey Juan Carlos
28933 Móstoles, Madrid
angel.velazquez@urjc.es, maximiliano.paredes@urjc.es
sergio.cavero@urjc.es, daniel.palacios@urjc.es

Resumen

Uno de los principales retos para la introducción de una materia obligatoria de informática en niveles educativos preuniversitarios es la falta de profesorado formado en informática. En nuestra universidad ofrecemos un máster para formar profesores en competencia digital y programación. La asignatura "Programación y Pensamiento Computacional I" presenta una introducción a la programación basada en bloques. En el curso académico 2021/22 se realizó un diseño de la asignatura basada en cuatro lenguajes de bloques en orden creciente de complejidad. Aunque los alumnos valoraron muy positivamente la asignatura, se identificaron varias cuestiones mejorables. En la comunicación se presentan los cambios introducidos durante el curso 2022/23, que consisten en la eliminación del lenguaje Code.org, una revisión de los apuntes de Scratch, el desarrollo de nuevos ejercicios de autoestudio para ScratchJr y Scratch, y la transición de Scratch a App Inventor. Se presentan los resultados obtenidos de rendimiento de los alumnos y de aceptación de la asignatura. La asignatura ha consolidado su aceptación por los alumnos, pero los cambios introducidos no han redundado en una mejora apreciable y aún persiste como reto el aprendizaje de los elementos más complejos, principalmente App Inventor.

Abstract

One of the main challenges to introduce informatics as a mandatory subject matter in pre-college education is the lack of teachers adequately trained on informatics. Our university offers master's studies aimed at teachers' development in digital competence and computer programming. The course "Programming and computational thinking I" introduces block-based programming. In the academic year 2021/22, the course was designed as a sequence of four languages, in increasing order of complexity. The students rated the course very high, but a few issues were amenable to improvement. In this paper, we present the changes introduced for the academic year 2022/23, comprising the removal of Code.org, re-elaboration of Scratch lecture notes, development of additional self-study exercises for ScratchJr and Scratch, and transition between Scratch and App Inventor. The paper also presents the outcomes obtained on students' performance and course acceptance. The course is consolidated according to students' high acceptance. However, the changes introduced did not produce a significant enhancement of acceptance, and learning the most complex elements remains an open challenge, especially App Inventor.

Palabras clave

Programación basada en bloques, ScratchJr, Scratch, App Inventor, Formación del profesorado.

1. Motivación

Existe un creciente interés en la enseñanza de la informática en los niveles preuniversitarios [5]. Uno de los retos que plantea es la formación de sus profesores, como reclaman sociedades científicas informáticas nacionales [14] e internacionales [1]. De momento el currículo escolar de España no sigue estos pasos, salvo algunos contenidos mínimos dentro de otras asignaturas, sobre todo de programación basada en bloques.

Actualmente, los maestros de Educación Infantil y Primaria se forman por caminos académicos distintos que los profesores de Educación Secundaria. Mientras que los primeros estudian grados específicos, los segundos han estudiado otros grados universitarios y

Este trabajo se ha financiado con el proyecto de investigación e-Madrid-CM (S2018/TCS-4307) de la Comunidad Autónoma de Madrid y los proyectos-puente PROGRAMA de la Universidad Juan Carlos (M2614 y M3035). El proyecto e-Madrid-CM también está financiado con los fondos estructurales FSE y FEDER.

completan su formación docente con el Máster de Formación del Profesorado. La formación específica en informática en los grados de Educación suele reducirse a la competencia digital o a un tema de programación con bloques, mientras que los graduados de ciencias o ingeniería pueden haber cursado alguna asignatura de programación.

Nuestra universidad ofrece desde el curso 2021/22 el "Máster en Competencia Digital y Pensamiento Computacional"[1], cuyo objetivo es proporcionar una formación en competencia digital y programación a profesores desde Educación Infantil hasta Bachillerato. Se ofrece en modalidad online y consta de una asignatura general sobre informática y educación, cuatro asignaturas de competencia digital y dos de programación.

En el curso 2021/22 se matricularon 20 alumnos, de los cuales uno causó baja. El perfil de los alumnos era heterogéneo: desde profesores sin experiencia docente a quien tenía 10 años de experiencia en FP de Grado Medio y Superior de la rama de Electrónica y Electricidad; desde profesores sin conocimientos de programación hasta graduados en Ingeniería Informática o en Videojuegos, abarcando todo tipo de grados. En el curso 2022/23 se matricularon 33 alumnos. A comienzo de curso se realizó una encuesta para conocer el perfil de los alumnos. Contestaron 26 alumnos, de los que 23 alumnos (88%) tenían una formación en Educación. Sin embargo, el curso pasado eran 7 alumnos sobre 20 (35%).

En una comunicación anterior presentamos en detalle una asignatura del máster, diseñada para el aprendizaje de la programación basada en bloques [10] y se identificaron algunos aspectos susceptibles de mejora. En esta comunicación presentamos algunas intervenciones didácticas realizadas para mejorar la asignatura. El apartado segundo da una breve visión de la asignatura, incluyendo los resultados de la valoración que los alumnos hicieron el curso 2021/22. En la siguiente sección se describen las intervenciones realizadas. Asimismo, el cuarto apartado presenta los resultados de aceptación obtenidos en este curso. Finalmente, se incluyen nuestras conclusiones.

2. Organización de la asignatura

La asignatura "Programación y Pensamiento Computacional I" debe proporcionar una base de programación mediante lenguajes de bloques [16] que facilite la iniciación posterior a la programación textual en la asignatura "Programación y Pensamiento Computacional II". Cada asignatura de programación tiene 6 créditos ECTS (equivalentes a 48 horas lectivas), impartiéndose en cuatrimestres consecutivos.

En el curso académico 2021/22, la asignatura "Programación y Pensamiento Computacional I" se estructuró en tres bloques:

1. Conceptos básicos de programación y Code.org.
2. Los lenguajes ScratchJr [6] y Scratch [12].
3. El lenguaje App Inventor [17].

A lo largo de la asignatura se presentaron los lenguajes en orden creciente de complejidad, con la siguiente distribución temporal:

- Code.org: 1 semana.
- ScratchJr: 2 semanas.
- Scratch: 3 semanas.
- App Inventor: 6 semanas.

El estudio de los lenguajes en orden creciente de dificultad no sólo pretendía facilitar su aprendizaje, sino que también permitía conocer una variedad de lenguajes, preparando a los alumnos para la enseñanza de la programación en cualquier etapa educativa preuniversitaria.

Los alumnos tenían a su disposición apuntes sobre los lenguajes, que incluían ejercicios a realizar, y tests de autoevaluación en la primera mitad de la asignatura. Se realizaron tutorías semanales para dudas mediante la plataforma Microsoft Teams. La evaluación de la asignatura se realizaba mediante tres prácticas y un examen final. La práctica de ScratchJr debía realizarse individualmente, mientras que las de Scratch y de App Inventor, en parejas. El examen constaba de una parte de teoría y otra de desarrollo o depuración de programas.

Se realizó una encuesta de satisfacción a los alumnos para que valoraran la asignatura. En términos generales, la asignatura tuvo una alta valoración. También se identificaron algunos aspectos que mejorar:

- Poca utilidad del entorno Code.org.
- Dificultad del lenguaje App Inventor.

3. Acciones de mejora de la asignatura

En el curso académico 2022/23 se realizaron varios cambios, que se explican en esta sección.

3.1. Code.org

En el anterior curso académico (2021/22), se había incorporado el lenguaje Code.org del "Curso express para pre-lectores" (orientado a niños de 4 a 8 años)[2] como primer lenguaje de bloques debido a su gran sencillez. El lenguaje tiene pocos tipos de bloques y se utiliza en un contexto limitado, que un personaje se mueva por un escenario para alcanzar un objetivo. Su

[1] https://www.urjc.es/estudios/master/4356-competencia-digital-y-pensamiento-computacional

[2] https://studio.code.org/s/pre-express-2021

gran sencillez facilita la presentación de conceptos básicos de programación tales como lenguaje de programación, instrucción (o bloque), secuencia, programa, bucle, edición, ejecución y depuración. Además, los cursos de Code.org tienen una gran difusión y su estética, retos y personajes son atractivos y entretenidos para niños y jóvenes.

Al analizar las valoraciones de los alumnos y planificar el curso académico 2022/23, se concluyó que podría suprimirse Code.org, dejando que ScratchJr desempeñara el mismo papel introductorio. La semana extra que se ganaba para Scratch permitía tratar elementos avanzados del lenguaje y, sobre todo, profundizar en sus elementos imperativos (variables y control condicional). Como estos elementos imperativos también existen en App Inventor, su conocimiento previo podría disminuir la dificultad de su aprendizaje, quedando limitada a otras dificultades.

Por tanto, se reestructuraron los bloques del temario y su duración:

1. El lenguaje ScratchJr: 2 semanas.
2. El lenguaje Scratch: 4 semanas.
3. El lenguaje App Inventor: 6 semanas.

3.2. ScratchJr

En principio, el tratamiento dado a ScratchJr apenas variaba. Sin embargo, la supresión de Code.org obligó a ampliar los apuntes de ScratchJr para explicar los conceptos básicos de programación antes comentados.

También se elaboraron unos apuntes con buenas prácticas de programación. De esta forma, los alumnos se familiarizarían con varios aspectos de la metodología de la programación que podrían ayudarles en sus prácticas. Se presentaban algunos elementos de buen estilo (selección de identificadores, bloques de terminación), de metodología de desarrollo de programas (desarrollo incremental) y se incluían patrones elementales [2] de programación (para simular algunos tipos de movimiento y la sincronización de dos o más personajes).

Por último, se amplió la colección de ejercicios de programación con ScratchJr a 36. En su mayoría son ejercicios de respuesta múltiple, cuya corrección es automática y facilita el envío inmediato a los alumnos de sus resultados. La colección de ejercicios permite seleccionar ejercicios para pruebas de autoestudio y exámenes, tanto en este curso como en el futuro. Se incluyeron dos pruebas de autoestudio, una por semana.

Se han analizado diversos tipos de ejercicios de programación [2, 11, 13] para identificar aquellos que puedan usarse con la programación basada en bloques. En general, la programación con bloques presenta características distintas de las de la programación textual imperativa.

En el caso de ScratchJr, la diferencia aún es mayor, ya que no existen variables. Por tanto, el programador solamente puede conocer el efecto de la ejecución de un programa observando sus efectos visibles (sobre personajes y páginas) o audibles. Otra consecuencia de la ausencia de variables en ScratchJr es que el concepto tradicional de algoritmo se diluye. Es cierto que pueden concebirse retos, pero no puede plantearse el diseño de un algoritmo como método preciso que permite trasformar unos datos en otros. A la inversa, el comportamiento de un trozo de código ya no corresponde a un problema resuelto, sino a acciones variadas de los personajes.

Los tipos de ejercicios finalmente desarrollados son los siguientes [15]:

- Ejercicios de conocimiento. Consisten en conocer hechos o características de la programación con ScratchJr, sin conllevar comprensión. Se desarrollaron 8 ejercicios de este tipo.
- Ejercicios predictivos y de rastreo. En estos ejercicios se pide, respectivamente, identificar el estado final de un personaje o la secuencia de estados por los que evoluciona la ejecución de un trozo de programa. En lenguajes textuales, este tipo de preguntas se basan en el valor final de las variables, el resultado de llamadas a función o en mensajes impresos. Dada la ausencia en ScratchJr de variables, funciones e instrucciones de salida, debe recurrirse a la emisión de sonidos o a características visuales de los personajes (si se mueve, si habla, su posición espacial, su orientación angular, su orientación a izquierda o derecha, su tamaño, si está visible o no, y su velocidad). Son ejercicios que exigen comprender el comportamiento de los bloques del lenguaje. Se desarrollaron 6 ejercicios (4 predictivos y 2 de rastreo).
- Ejercicios de comprensión. Estos ejercicios requieren una comprensión del efecto global de los bloques del programa. Normalmente se pide identificar programas con un comportamiento equivalente al de un programa dado. Se desarrollaron 8 ejercicios de comprensión. La Figura 1(a) muestra un ejemplo de ejercicio de comprensión, con varias respuestas correctas (opciones 1 y 3).
- Ejercicios de traducción. Se trata de ejercicios con un enunciado en lenguaje natural que describe el comportamiento de uno o varios personajes, pidiéndose su traducción al lenguaje de programación. Son ejercicios que requieren comprender el comportamiento de los bloques del lenguaje. Algunos bloques son sencillos de identificar, como los bloques de movimiento. Sin embargo, no es tan sencillo identificar la necesidad de usar algunos bloques de control ni la forma en que deben usarse. Se desarrollaron 14 ejercicios de traducción. La Figura 1(b) muestra un ejemplo de pregunta de este tipo.

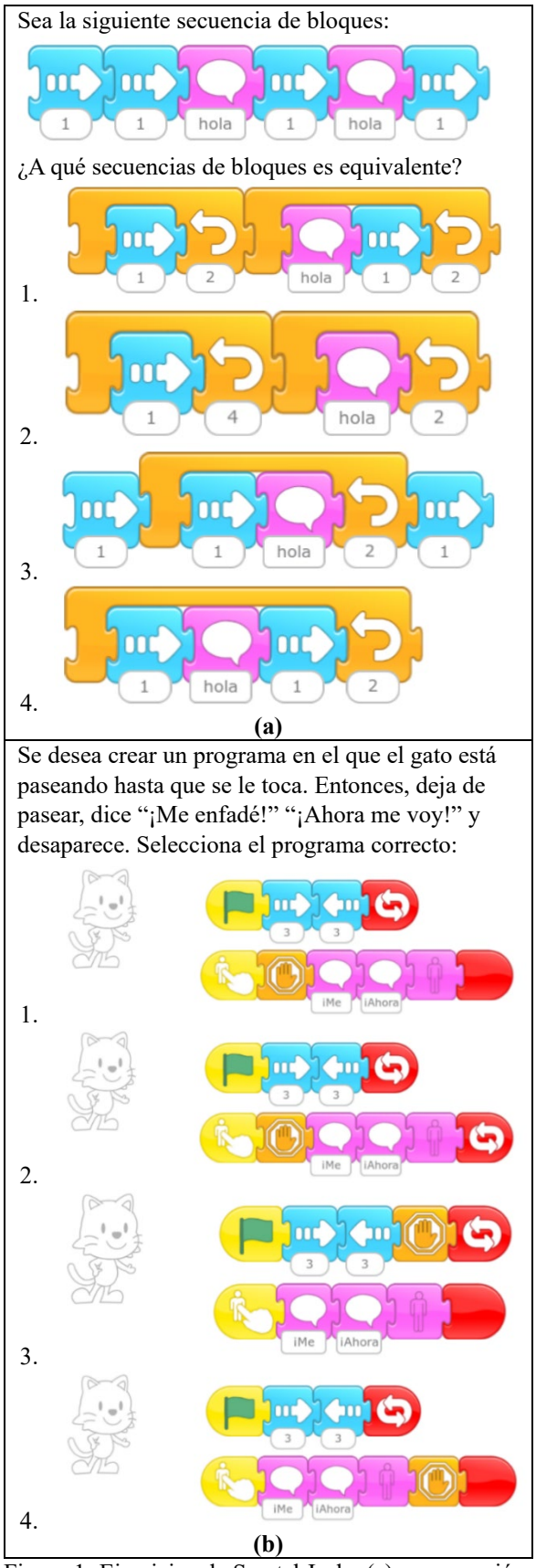


Figura 1: Ejercicios de ScratchJr de: (a) comprensión, y (b) traducción.

3.3. Scratch

Ya hemos comentado que el bloque de Scratch se alargó de 3 a 4 semanas. Esta ampliación permitía presentar elementos avanzados que apenas se estudiaron el curso anterior por falta de tiempo y dedicar más tiempo a los elementos imperativos de Scratch, como variables y control condicional. Como estos elementos también existen en App Inventor, su conocimiento previo podría reducir parte de la dificultad del lenguaje (aunque otras dificultades persistieran). Asimismo, se desarrollaron apuntes con consejos prácticos y se revisó y amplió la colección de ejercicios. Veamos cada medida.

El reparto de los elementos de Scratch entre las cuatro semanas quedó como sigue:

1. Conceptos básicos, entorno Scratch, secuencias, movimientos, bucles.
2. Apariencia y sonido, eventos, valores y expresiones.
3. Variables, control basado en condiciones, algoritmos.
4. Listas, otros bloques de control, bloques nuevos.

Ya existían apuntes del curso académico pasado para las tres primeras semanas. Sin embargo, se ampliaron los apuntes de las partes segunda y tercera para facilitar la transición a la programación imperativa y App Inventor. Así, en los apuntes de la segunda semana se detalló cómo se evalúa, paso a paso, una expresión. Asimismo, en la parte tercera, se detalló el comportamiento de las conectivas lógicas y se resaltó la importancia de una adecuada secuenciación en las instrucciones con variables.

La inclusión de una cuarta parte obligó a la preparación de apuntes para los nuevos elementos. Se dedicó especial atención a listas, clones y bloques nuevos. La definición de bloques nuevos permitió realizar una pequeña introducción a la recursividad.

Cabe destacar que se añadió una breve introducción a los algoritmos, por su importante relación con la programación textual y por la creciente popularidad del término. En los apuntes del módulo tercero, se introdujeron los conceptos de problema y algoritmo, incluyendo varios ejemplos (máximo de dos números, cálculo del factorial de un número, cálculo del enésimo elemento de la serie de Fibonacci y suma de una serie aritmética). En los apuntes del módulo cuarto, los bloques nuevos permitieron crear una interfaz que permitía (dentro de las limitaciones de Scratch) encapsular cada algoritmo dentro de una definición de bloque adecuada al problema correspondiente.

La recursividad se ilustró con la definición matemática del factorial. Se mencionaron los algoritmos recursivos, pero no se llegó a codificar ninguno en Scratch, dada su excesiva dificultad para los alumnos sin experiencia previa en programación (exigía el manejo ex-

plícito de una lista como pila de control de la recursividad). Sin embargo, se presentó la creación recursiva de clones, que es visible y resulta muy llamativa.

Al igual que con ScratchJr, se prepararon apuntes de consejos prácticos. En este caso se presentaron algunos patrones elementales con expresiones, variables y listas, así como consejos didácticos y referencias a materiales para la docencia de Scratch.

Por último, se hizo un esfuerzo de ampliación de la colección de ejercicios. Los ejercicios corresponden a las mismas categorías que para ScratchJr [15]: de conocimiento (10 ejercicios), de comprensión (14 ejercicios), predictivos o de rastreo (17), y de traducción (7).

Obsérvese que, al existir variables en Scratch, pueden plantearse ejercicios comunes en programación textual, como que se resuma el efecto de un algoritmo. De hecho, la mayor parte de los ejercicios predictivos desarrollados son sobre variables o listas (10 de 17).

3.4. App Inventor

Para facilitar la transición de Scratch a App Inventor, se elaboró una tabla que mostraba la correspondencia entre elementos de ambos lenguajes. Se incluyeron variables, eventos, distintos bloques de control, procedimientos y funciones.

La correspondencia busca orientar a los alumnos. En algunos casos la correspondencia es casi total, como en variables y eventos, mientras que en otras sólo cumplen funciones parecidas, como el caso del bucle "repetir hasta que" de Scratch y el "*while test do*" de App Inventor. Hay casos en los que elementos de un lenguaje no tienen correspondencia en el otro, como los clones de Scratch o las funciones de App Inventor.

La tabla estaba accesible para los estudiantes en la plataforma virtual. Además, se anunció en el foro y se indicó que sería conveniente consultarla al inicio de cada bloque nuevo de App Inventor que estudiasen.

También se hizo algún cambio menor en el bloque de App Inventor para facilitar el entendimiento del enunciado de la práctica, mejorando la narrativa del problema a resolver y aclarando la rúbrica y los entregables de la misma.

4. Evaluación

Se recogió la opinión de los alumnos sobre la asignatura mediante una encuesta. En concreto, se quería conocer su percepción de la dificultad y utilidad de los distintos lenguajes de bloques, así como de la utilidad de los recursos docentes y herramientas usadas en la asignatura.

4.1. Instrumento, muestra y análisis

El cuestionario constó de las siguientes partes:

1. Recursos docentes: utilidad de los recursos proporcionados (apuntes, diapositivas, vídeos explicativos, tutorías y ejercicios).
2. Herramientas de programación: utilidad de los entornos de desarrollo usados para aprender cada lenguaje de programación (ScratchJr, Scratch y App Inventor).
3. Lenguajes de programación: dificultad de cada lenguaje.
4. Transición de Scratch a App Inventor: valoración de la tabla de equivalencias.
5. Sistema de evaluación: aceptación del sistema de evaluación utilizado.
6. Preguntas abiertas: aspectos más difíciles y opinión global sobre la asignatura.

En las cuatro primeras partes del cuestionario, se utilizaron preguntas con una escala Likert de 5 valores, desde 1 ("poco o nada"), pasando por una valoración intermedia de 3 ("aceptable") hasta 5 ("mucho"). Para el sistema de evaluación, se preguntó si les parecía adecuado (sí/no) y una pregunta abierta sobre cómo lo mejorarían. Por último, para recabar la opinión del estudiante se formularon dos preguntas abiertas.

En las tres primeras partes del cuestionario se distinguió entre los bloques I y II frente al bloque III. Las dos últimas partes del cuestionario valoraban aspectos globales de la asignatura.

El cuestionario era igual al del curso académico 2021/22, excepto que se suprimió la valoración de Code.org y se incluyó la valoración de la tabla de equivalencias entre Scratch y App Inventor.

La encuesta se suministró en el aula virtual antes de realizar el examen. Se explicó a los alumnos el objetivo del cuestionario y su carácter voluntario. Los alumnos debían marcar su consentimiento. También era voluntario que se identificaran con su nombre. Hubo 31 respuestas, de las que 8 fueron anónimas.

La información recogida de las respuestas cerradas se analizó mediante estadística descriptiva usando IBM SPSS Statistics 27, mientras que las preguntas abiertas se analizaron con métodos cualitativos.

4.2. Resultados cuantitativos

El Cuadro 1 muestra algunos estadísticos descriptivos de las variables que miden la percepción de la utilidad de los entornos de programación y la dificultad de los lenguajes. También se muestra la percepción de utilidad de la tabla de correspondencia entre Scratch y App Inventor. La mediana para los entornos muestra que ScratchJr y Scratch son considerados muy útiles y App Inventor, útil. La tabla de correspondencia ha sido considerada útil. En cuanto a la dificultad de los lenguajes, App Inventor es percibido como muy difícil mientras que los lenguajes de Scratch son percibidos de dificultad media.

La Figura 2 muestra la comparativa de estas percepciones entre los cursos 2021/22 y 2022/23. Se puede observar que las valoraciones del curso pasado son casi iguales para ScratchJr, empeorando algo para Scratch y más para App Inventor.

Variable	Media	Mediana	Desv. Típica
Util_ScratchJr	4,39	5	0,844
Util_Scrath	4,61	5	0,615
Util_AppInv	3,81	4	1,276
Util_tabla	3,71	4	0,902
Dificil_ScratchJr	2,45	3	1,15
Dificil_Scratch	3,32	3	1,249
Dificil_AppIn	4,35	5	0,915

Cuadro 1: Estadística descriptiva de las variables de percepción de utilidad y dificultad (n=31).

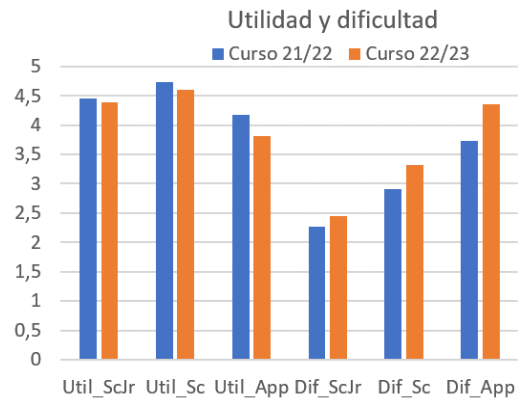
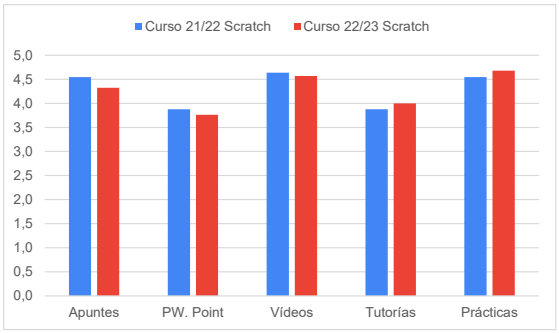
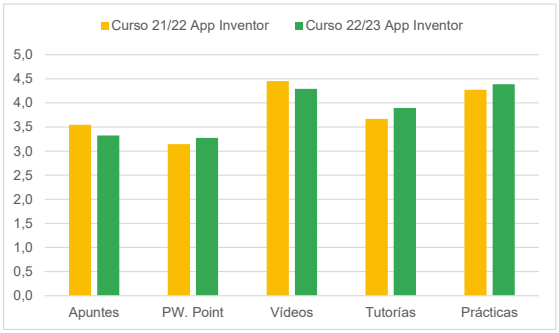


Figura 2: Valoración de la percepción de utilidad y dificultad por curso.



(a)



(b)

Figura 3: Valoración de los recursos docentes por curso para: (a) ScratchJr y Scratch, y (b) App Inventor.

La Figura 3 muestra la comparación por cursos de la valoración media que los estudiantes hicieron de los recursos utilizados para estudiar, como apuntes facilitados por el profesor, presentaciones, etc. Se puede ver que la tendencia es similar en ambos cursos académicos. La valoración por los estudiantes del curso 2022/23 de los apuntes y vídeos ha disminuido ligeramente en relación con el curso 2021/22, mientras que el uso de tutorías y la realización de ejercicios ha obtenido una valoración ligeramente mejor en el curso 2022/23 que en el curso anterior.

En cuanto al sistema de evaluación usado en la asignatura, 27 alumnos estuvieron de acuerdo (87%) y 4 no (13%).

En lo relativo a los resultados académicos de los estudiantes de ambos cursos, los estudiantes del curso 2021/22 obtuvieron una calificación final media de 7'46 sobre 10, mientras que los del curso 2022/23 obtuvieron 7'76 puntos. Por tanto, se ha producido una ligera mejora.

4.3. Resultados cualitativos

El cuestionario incluía tres preguntas abiertas, cuyo análisis presentamos en esta sección.

Aspectos más difíciles de aprender

Un total de 28 alumnos contestaron a la pregunta de qué partes de la asignatura les parecían más difíciles. Tres de ellos dieron una respuesta doble. Por tanto, se recogieron un total de 31 respuestas simples. Corresponden a los siguientes aspectos, presentados en orden decreciente de frecuencia:

- App Inventor: 20 respuestas. De ellas, 13 respuestas eran sobre el propio App Inventor; a veces se daban detalles específicos: proceso de desarrollo (cinco respuestas), *spinners* (dos), variables globales. A su vez, 7 respuestas eran sobre cuestiones docentes; estas opiniones se dividían entre quienes pedían mayor apoyo para realizar la práctica (cuatro respuestas) o mejores apuntes (tres).
- Programar en general, sin identificar ningún lenguaje: 7 respuestas. En la mitad de los casos reconocían su completa inexperiencia previa con programación. Algunas respuestas señalaban aspectos específicos: variables, las "partes más matemáticas", enunciados de preguntas, funcionamiento de los bloques, listas, bloques condicionales.
- Scratch: 4 respuestas. Identificaban partes concretas: variables (dos respuestas), operadores, clones, bloques nuevos, algoritmos, los aspectos "más matemáticos".

Aunque se preguntaba por las dificultades encontradas, una respuesta daba su opinión positiva sobre el orden creciente de dificultad de los lenguajes seguidos en la asignatura.

Sistema de evaluación

Se recogieron 12 respuestas, de las cuales 7 sugerían cambios en la evaluación de la asignatura.

Cinco alumnos sugerían que se eliminaran las preguntas de desarrollo de programas presentes en el examen e incluso el propio examen. Otra alumna proponía más actividades prácticas. Finalmente, otra encontraba demasiada diferencia entre las actividades del aula y la práctica de App Inventor.

Opinión sobre la asignatura

Dieron su opinión 30 de los 31 alumnos que contestaron el cuestionario. Algunas opiniones fueron positivas, otras negativas y, en la mitad de las contestaciones, aparecían mezcladas opiniones de ambos signos.

Las opiniones positivas pueden agruparse en dos categorías:

- Asignatura interesante: 15 respuestas. Algunas son bastante entusiastas, por ejemplo, "*La asignatura es la más atractiva del máster con diferencia. Me da igual que influya o no en la nota, es por dejarlo claro. Se pueden hacer muchos proyectos interesantes gracias a los programas con los que hemos trabajado y, yo por lo menos, he aprendido cosas que desconocía totalmente*".
- Asignatura útil en el aula: 11 respuestas. Sin embargo, una alumna matizaba su opinión: "*La parte de Scratch (tanto Jr como el original) me ha parecido muy útil de cara a desarrollarlo como docente en el colegio. Sin embargo, App Inventor no lo veo tan útil para un nivel de Educación Primaria*".

Las opiniones negativas o mixtas pueden considerarse de tres tipos:

- Método instruccional: 9 respuestas. Realizaban sugerencias variadas: más videotutoriales (tres respuestas), grabación de las sesiones de dudas (tres), apuntes (dos), resolver más prácticas (una).
- Contenidos de la asignatura: 5 respuestas. Tres comentarios indicaban que solamente les servían algunos lenguajes para la etapa educativa en la que dan clase, mientras que otros dos indicaron que el contenido de la asignatura era excesivo.
- Trabajo en pareja: 2 respuestas. Expresaban dos opiniones opuestas sobre el trabajo en pareja.

5. Discusión

Los resultados cuantitativos indican que la percepción de la utilidad de los entornos de desarrollo es alta o muy alta, igual que la complejidad percibida de los lenguajes es media para ScratchJr y Scratch, pero muy alta para App Inventor. Asimismo, los materiales docentes elaborados reciben una valoración muy alta en el caso de vídeos, ejercicios, así como los apuntes de ScratchJr y Scratch, siendo alta para el resto de materiales. Finalmente, el sistema de evaluación usado en la asignatura es valorado positivamente por una inmensa mayoría.

Si comparamos los resultados obtenidos con los del curso anterior, nos ha sorprendido que ha empeorado ligeramente la percepción de los estudiantes, de forma proporcional a la complejidad de cada lenguaje, a pesar de las modificaciones realizadas en el actual curso 2022/23. Una posible explicación puede residir en el diferente perfil de los alumnos matriculados en ambos años.

El lenguaje de App Inventor sigue siendo muy complejo para los estudiantes, por lo que debemos centrar los esfuerzos en mejorar la propia parte de App Inventor. Por ejemplo, podrían adaptarse los ejemplos prácticos que se enseñan en los vídeos a unos ejemplos más sencillos y conceptuales, incluso podría analizarse si se pueden utilizar los mismos ejemplos que en Scratch. También conviene revisar el uso de la tabla de correspondencia entre Scratch y App Inventor y el tratamiento de los elementos imperativos de Scratch. La repetición de diversos elementos en Scratch y App Inventor sería una forma limitada de currículo en espiral [4]. De hecho, se mostrarían los aspectos comunes de elementos de programación presentes en los dos lenguajes, así como sus diferencias, como propugna la teoría de la variación [9].

Del análisis de las respuestas abiertas, se deduce que los alumnos encuentran la asignatura interesante personalmente y útil para su labor docente, incluso para los que no tenían ningún conocimiento previo de programación. Las citas incluidas en la sección anterior son ilustrativas; existen muchas con el mismo tono, por ejemplo, "*me encanta*", "*me ha gustado mucho*", "*he aprendido cosas muy prácticas y ya lo estoy usando con los niños*".

No obstante, los alumnos piensan que algunos aspectos de la asignatura deberían mejorarse. Convendría hacerse un esfuerzo en mejorar los apuntes de App Inventor, especialmente para ayudar a comprender el proceso de desarrollo y para orientar la práctica. También existen comentarios sobre la mayor dificultad de los elementos más avanzados, matemáticos o algorítmicos de Scratch. Por tanto, en general conviene revisar la parte de programación de ambos lenguajes que contiene elementos imperativos más abstractos, como las variables.

Por otro lado, al ofrecerse el máster en modalidad online, se pide la grabación de las tutorías. Actualmente son tutorías de dudas, pero podrían incluir algo de teoría, al menos una visión panorámica. De igual modo, podrían incluirse más videotutoriales.

En cuanto a la evaluación de la asignatura, algunos alumnos reclaman la supresión de la parte práctica del examen. Este aspecto puede revisarse, siempre dentro de las horquillas que la memoria acreditada del título

establece para cada actividad de evaluación sobre la nota final.

6. Conclusiones

Hemos presentado el estado de una asignatura de introducción a la programación basada en bloques para profesores de etapas preuniversitarias tras impartirla en dos cursos académicos. Consideramos que la asignatura se encuentra asentada y resulta satisfactoria (incluso, para algunos alumnos, es la más atractiva del máster). Sin embargo, las críticas y sugerencias de los alumnos permiten identificar mejoras en aspectos concretos de la asignatura, como apuntes, videotutoriales, tutorías online y sistema de evaluación. En general, conviene centrar los esfuerzos en App Inventor, aunque también deben revisarse algunos elementos de Scratch.

Los autores consideran que hay margen para realizar algunas mejoras, de forma coherente con la literatura educativa. Así, conviene ampliar la colección de ejercicios de autoestudio para que los alumnos interesados puedan comprobar sus conocimientos y ejercitarse. Un aspecto clave para aprender de los errores es disponer de una realimentación informativa [7]. Actualmente, son muy escuetos o no existen mensajes de realimentación de los ejercicios, por lo que sería conveniente revisarlos y completarlos. Asimismo, sería interesante ampliar algunos aspectos ya introducidos en la asignatura, como patrones elementales [2]. Una última mejora podría residir en adoptar fomentar metodologías de aprendizaje más activas, como "usar-modificar-crear" [8]. Se trata de la mejora más difícil de desarrollar, dado apretado del temario, pero podemos analizarla cuando revisemos el método de evaluación.

Referencias

[1] ACM Europe & Informatics Europe. *Informatics for All: The Strategy*. 2018. Disponible en: https://www.informaticsforall.org/informatics-for-all-the-strategy/.

[2] Kashif Amanullah y Tim Bell. Teaching resources for young programmers: The use of patterns. En *Proc. 2020 IEEE Global Engineering Education Conf., EDUCON 2020*, Alberto Cardoso, Gustavo R. Alves y Teresa Restivo (eds.), pp. 679-687, 2020.

[3] Matt Bower. A taxonomy of task types in computing. En *Proc. 13th Annual Conf. Innovation and Technology in Computer Science Education, ITiCSE 2008*, pp. 281-285.

[4] Jerome S. Bruner. *The Process of Education*. Harvard University Press, 1960.

[5] The Committee on European Computing Education (CECE), Informatics Europe & ACM Europe. *Informatics Education in Europe: Are We All in the same Boat?* 2017. DOI: 10.1145/3106077.

[6] Louise P. Flannery, Elisabeth R. Kazakoff, Paula Bontá, Brian Silverman, Marina U. Bers y Mitchel Resnick. Designing ScratchJr: Support for early childhood learning through computer programming. En *Proc. 12th International Conf. Interaction Design and Children, IDC '13*, pp. 1-10.

[7] Diana Laurillard. *Teaching as a Design Science*. Routledge, 2012.

[8] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith y Linda Werner. Computational thinking for youth in practice. *Inroads*, 2(1):32-37, 2011.

[9] Ference Marton y Amy B.M. Tsui. *Classroom Discourse and the Space of Learning*. Routledge, 2004.

[10] Maximiliano Paredes Velasco y J. Ángel Velázquez Iturbide. Una asignatura para la formación del profesorado en programación mediante lenguajes basados en bloques. *Actas de las JENUI 2022*, 7:337-344. Disponible en https://aenui.org/actas/pdf/JENUI_2022.pdf.

[11] Noa Ragonis. Type of questions – The case of Computer Science. *Olympiads in Informatics* 6:115-132, 2012.

[12] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman e Yasmin Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[13] Alexander Ruf, Marc Berges y Peter Hubwieser. Classification of programming tasks according to required skills and knowledge representation. En *Proc. 8th International Conf. Informatics in Schools, ISSEP 2015*, A. Brodnik y Jan Vahrenhold (eds.). Springer, LNCS 9378, pp. 57–68.

[14] SCIE y CODDII. *Informe del grupo de trabajo SCIE/CODDII sobre la enseñanza preuniversitaria de la informática*. 2018. Disponible en: https://www.scie.es/actividades/educacion/.

[15] J. Ángel Velázquez Iturbide. Designing exercises for block-based languages: The case of ScratchJr. En *Proc. 10th International Conference Technological Ecosystems for Enhancing Multiculturality, TEEM 2022*, F.J. García Peñalvo y A. García Holgado (eds.). Springer, LNET, 2023.

[16] David Weintrop, Block-based programming in computer science education. *Communications of the ACM*, 62(8):22-25, agosto 2019.

[17] David Wolber, Hal Abelson, Ellen Spertus y Liz Looney. *App Inventor: Create Your Own Android Apps*. O'Reilly Media, 2011.