Actas de las Jenui, vol. 7. 2022.
Páginas: 343-350

Una asignatura para la formación del profesorado en programación mediante lenguajes basados en bloques

Maximiliano Paredes Velasco
Departamento de Informática y Estadística
Universidad Rey Juan Carlos
28933 Móstoles, Madrid
maximiliano.paredes@urjc.es

J. Ángel Velázquez Iturbide
Departamento de Informática y Estadística
Universidad Rey Juan Carlos
28933 Móstoles, Madrid
angel.velazquez@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 mínimamente formado en informática, y de forma destacada en programación. En nuestra universidad hemos iniciado la impartición de un máster orientado a la formación de profesores en competencia digital y programación. La comunicación presenta la asignatura "Programación y Pensamiento Computacional I", y su enfoque como introducción al aprendizaje de la programación. La asignatura se centra en lenguajes basados en bloques dada su menor dificultad de aprendizaje y su interés en etapas preuniversitarias. La asignatura muestra varios lenguajes basados en bloques en orden creciente de complejidad: Code.org, ScratchJr, Scratch y App Inventor. Se presentan los objetivos, contenidos, ejercicios y pruebas de evaluación de la asignatura, así como la opinión de los alumnos sobre la asignatura. Finalmente, se presentan las conclusiones de los autores tras el primer año de impartición y las líneas de mejora previstas.

Abstract

One of the main challenges to introduce "informatics" as a mandatory subject matter in pre-college education is the lack of teachers with a minimum background on informatics, in particular on programming. This academic course, we have offered for the first time at our university a novel master aimed at educating teachers in digital competency and programming. The paper presents the course "Programming and Computational Thinking I", which is conceived as an introduction to programming. The course is focused on block-based languages, given their lower learning difficulty and their interest in pre-college educational stages. The course shows several block-based languages in increasing order of complexity: Code.org, ScratchJr, Scratch and App Inventor. The paper presents the goals, syllabus, exercises and assessment tests of the course, as well as students' opinion on the course. Finally, we present the instructors' conclusions based on their experience and futures lines of improvement.

Palabras clave

Programación basada en bloques, ScratchJr, Scratch, App Inventor, contenidos, ejercicios, opinión de los alumnos.

1. Introducción

En la última década ha aparecido una generación de lenguajes de programación basados en bloques, de los que el más conocido es Scratch [15]. Estos lenguajes eliminan algunas dificultades para su aprendizaje [4], siendo incluso una alternativa para la introducción a la programación en la universidad [20], aunque su uso no esté exento de problemas [13].

Los lenguajes basados en bloques se usan principalmente en la educación preuniversitaria, con niños y jóvenes. Su uso coincide con la implantación en numerosos países de estudios de informática en niveles preuniversitarios [6]. Uno de los retos que plantea esta nueva situación es una buena formación de los futuros maestros, sobre lo que han llamado la atención repetidamente sociedades científicas informáticas nacionales [16] e internacionales [10]. Aunque nuestro país no sigue de momento estos pasos, parece previsible que terminará haciéndolo, dada la creciente importancia de la informática en todos los ámbitos de nuestra sociedad. Por tanto, no deberíamos perder el tren de investigar la didáctica de la informática para profesores.

En España, los profesores de Educación Infantil y Primaria se forman por caminos académicos distintos que los profesores de Educación Secundaria. Mientras

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

que los primeros estudian grados específicos, los segundos han estudiado otros grados universitarios y completan su formación docente con el Máster de Formación del Profesorado. La formación específica en informática puede reducirse a un tema de programación con bloques en alguna asignatura de los grados de Educación, mientras que los graduados de ciencias o ingeniería pueden haber cursado asignaturas de programación. El Máster también puede incluir contenidos más extensos en la especialidad de Tecnología.

La comunicación presenta una asignatura de máster diseñada para el aprendizaje de la programación basada en bloques para profesores en formación o en servicio. En el apartado siguiente se presenta la organización general de la asignatura. Los dos apartados posteriores presentan en detalle las dos mitades de la asignatura. El quinto apartado presenta los resultados de una encuesta de opinión realizada entre los alumnos. Finalmente se incluye una reflexión y las conclusiones.

2. Organización de la asignatura

El "Máster en Competencia Digital y Pensamiento Computacional"[1] está concebido para proporcionar una formación en competencia digital y programación a profesores de cualquier etapa educativa, desde Educación Infantil hasta Bachillerato. Se ofrece en modalidad online. Consta de una asignatura general sobre informática y educación, cuatro asignaturas de competencia digital y dos de programación. Las asignaturas de competencia digital cubren las cinco áreas competenciales del marco DigComp [9].

Cada asignatura de programación tiene 6 créditos ECTS, equivalente a una asignatura presencial de 4 horas de clase semanales. Las dos asignaturas se imparten secuencialmente en dos cuatrimestres. La asignatura "Programación y Pensamiento Computacional I" debe proporcionar una base de programación mediante lenguajes con bloques que facilite la iniciación posterior a la programación textual en la asignatura "Programación y Pensamiento Computacional II".

En esta comunicación presentamos la primera asignatura, que consta de tres bloques:

I. Conceptos básicos de programación y Code.org.
II. Los lenguajes ScratchJr [7] y Scratch [15].
III. El lenguaje App Inventor [21].

A lo largo de la asignatura se van introduciendo los cuatro lenguajes mencionados en orden creciente de complejidad y 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 facilita su aprendizaje, sino que permite conocer una variedad de lenguajes, preparando a los alumnos para la enseñanza de la programación en cualquier etapa educativa preuniversitaria.

La asignatura se evalúa mediante tres tipos de tareas: prácticas (60% de la nota final), examen final (35%) y otras actividades (5%). Las prácticas consisten en el desarrollo de programas y se describen en las siguientes secciones. El examen final consta de preguntas multiopción con una sola opción correcta y ejercicios de desarrollo. Las restantes actividades han sido tests con ejercicios de comprensión y trabajos de ampliaciones puntuales de ejercicios resueltos por el profesor. Dado que el máster se imparte en modalidad online, todos los materiales docentes y pruebas de evaluación se encuentran disponibles en el aula virtual. El examen, sin embargo, es presencial, aunque se realizó y entregó por medio del aula virtual.

En el curso 2021-22 se matricularon 20 alumnos, de los cuales uno causó baja. De los 19 alumnos, 15 aprobaron con notas variadas, 1 suspendió (no se presentó al examen aunque había realizado las prácticas) y 3 no presentaron ninguna actividad.

El perfil de los alumnos era heterogéneo, sin que predominara ninguno: 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, incluyendo Educación Infantil o Primaria.

3. Bloques I y II: Code.org, ScratchJr y Scratch

3.1. Contenidos

Ambos bloques contienen un vídeo corto de presentación de cada lenguaje. Asimismo, se proporcionan apuntes y ejercicios de autoevaluación (tipo test), así como materiales complementarios para aquellos alumnos que quieran profundizar. Los apuntes de cada parte incluyen una descripción verbal de las reglas de ejecución de los elementos más destacados del lenguaje, así como una identificación de la información que caracteriza el estado de la ejecución de un programa [18]. También incluyen una recapitulación final, con los elementos de programación presentados (conceptos, actividades y herramientas).

El bloque I sirve de introducción a algunos conceptos básicos de programación usando un lenguaje basado en bloques muy simple. Se ha seleccionado parte del curso "Pre-reader Express", que Code.org ofrece para niños de 4 a 8 años. Las lecciones seleccionadas

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

permiten que los alumnos se familiaricen con el manejo de bloques y algunos conceptos básicos, como programa, edición, secuencia, bucle y ejecución. Como materiales complementarios para los alumnos, se presenta el informe SCIE-CODDII de enseñanza preuniversitaria de la informática [16] y un análisis del ambiguo término "pensamiento computacional" [19].

El bloque II consta de dos temas, dedicados a otros dos lenguajes basados en bloques; en orden de complejidad son ScratchJr y Scratch. Con ScratchJr se presentan otros conceptos de programación: paralelismo, eventos, sincronización, bucles infinitos, ámbito y algunos otros bloques de control. Como material complementario se proporcionan materiales disponibles en la web de ScratchJr[2] sobre los bloques del lenguaje, el editor gráfico y "tarjetas" (programas de ejemplo).

El lenguaje Scratch se presenta gradualmente en tres partes, dedicadas a: (a) visión general del lenguaje, secuencias, movimientos y bucles; (b) apariencia y sonido, eventos, valores y expresiones, y (c) variables y bloques condicionales. Para cada parte, junto a apuntes y un test, se proporciona el código fuente de algunos programas Scratch. Como material complementario, se aportan tarjetas de Scratch disponibles en la web del lenguaje[3], la guía de "Informática Creativa" [5] y un libro electrónico sobre programación Scratch [3].

3.2. Tareas

Como se ha comentado en el apartado 2, la evaluación de los alumnos se basa en tests y prácticas, aparte del examen final. Se aprovecharon las tareas de evaluación para presentar a los alumnos una introducción a la didáctica de la programación. Por un lado, se explicó la tipología de ejercicios de programación existentes en los tests. Además, se pidió que el programa desarrollado en cada práctica se situara en algún contexto educativo que debían identificar en la memoria.

Se incluyeron tres tests:

- Conceptos básicos y Code.org: 3 preguntas de conocimiento (en términos de la taxonomía de Bloom [2]).
- ScratchJr: 10 preguntas de conocimiento, de comprensión, predictivos [12], de rastreo y de traducción de texto a código.
- Scratch: 12 preguntas de los mismos tipos que para el lenguaje anterior.

También se propusieron dos prácticas consecutivas. En ambos casos se pedía desarrollar un programa:

- ScratchJr. El programa debía tener entre 2 y 4 páginas de ScratchJr. Su realización era individual. Debían presentar un documento con tres apartados: breve explicación de qué hace el programa, capturas de pantalla del programa desarrollado y comentarios sobre la práctica.
- Scratch. El programa debía contener varios personajes que interactuaran entre sí o con el usuario. Dada la mayor complejidad del lenguaje, esta práctica debía realizarse en equipos de 2 alumnos, creados por el profesor con perfiles similares o complementarios. No obstante, algunas incidencias obligaron a ajustar algunos grupos, que quedaron finalmente con 1 ó 3 miembros. Cada equipo debía entregar un fichero comprimido que contuviera el programa Scratch y un informe con una estructura parecida a la anterior.

Se indicó en ambas prácticas que se calificarían la calidad y claridad del programa y de la memoria explicativa. En la práctica de Scratch también se tendría en cuenta, aunque en menor medida, la complejidad técnica del programa.

4. Bloque III: App Inventor

4.1. Contenidos

Los contenidos de este tercer y último bloque están organizados en 5 temas. En el Tema 1 se estudian las bases para desarrollar aplicaciones para móviles Android con App Inventor. Se da a conocer el entorno de desarrollo, explicando la creación de proyectos y las pantallas para diseñar interfaces de usuario y para programar los bloques. Posteriormente se explica la generación y descarga del fichero .APK a partir de los proyectos creados para su ejecución en teléfonos Android. Se explica también cómo validar las aplicaciones desarrolladas mediante la ejecución en dispositivos reales con AI Companion o mediante la instalación del emulador Android en un ordenador personal.

Los aspectos anteriores son críticos ya que constituyen un cambio de paradigma de diseño considerable para los estudiantes. Con los lenguajes ScratchJr y Scratch, el diseño de la interacción de la aplicación está combinado y se presenta conjuntamente con la programación de su comportamiento. Sin embargo, App Inventor fuerza a separar estos dos aspectos y enfocar la creación de aplicaciones desde un punto de vista más profesional y con mayor complejidad técnica. El estudiante tiene que hacer una primera fase de diseño de la interfaz en una pantalla específica (ver Figura 1), donde se diseñan los recursos gráficos, y después se pasa a una segunda pantalla del entorno de desarrollo donde el estudiante programa el comportamiento de la interfaz desarrollada (ver Figura 2).

En el Tema 2 se estudia la captura y tratamiento de eventos de los principales *widgets* de interfaces de usuario, como son botones, *sprites*, *radio-buttons*, etc. En los Temas 3 y 4 se muestran los principales bloques

[2] https://www.scratchjr.org/

[3] https://resources.scratch.mit.edu/www/cards/es/scratch-cards-all.pdf

de control del flujo de ejecución: bloques condicionales y de repetición. De los bloques condicionales, se estudia la estructura *if_then_else* y sus variantes. En los bloques de repetición, se trabajan las estructuras *for_each* (en diferentes variantes) y *while*. Junto con los bloques de repetición, se estudian los bloques *Lists* como tratamiento de estructura de datos lineal. Por último, en el Tema 5 se trabaja la subprogramación mediante los bloques *Procedures* y bloques *call* y el manejo y procesamiento de algunos sensores del teléfono móvil, como el sensor de orientación o el reloj interno.

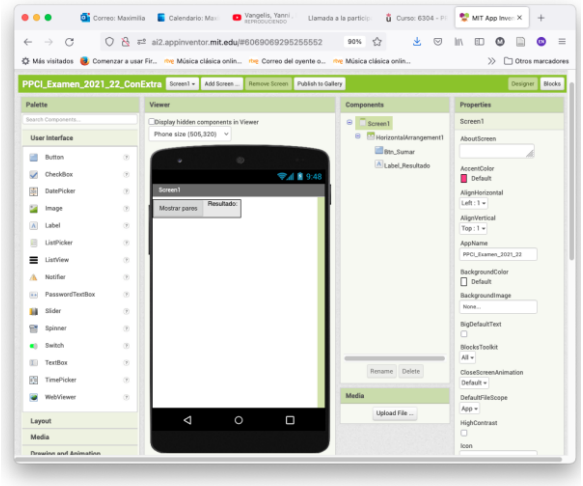

Figura 1: Fase de diseño de la interfaz de usuario.

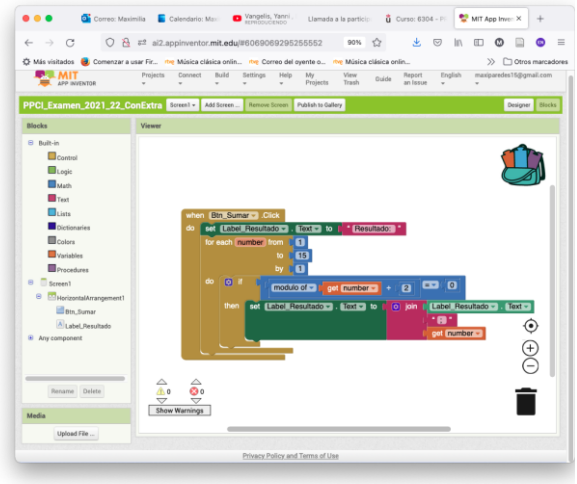

Figura 2: Fase de programación con bloques.

4.2. Tareas

La evaluación del bloque de App Inventor se realizó de manera continuada mediante la entrega de cuatro tareas desarrolladas progresivamente durante la segunda mitad del cuatrimestre. Las tres primeras tareas tenían carácter voluntario y su realización era individual (tareas T1-T3). Sin embargo, la tarea T4 era obligatoria y fue realizada por las parejas (o grupos de tres estudiantes en casos excepcionales) ya constituidas en la primera parte de la asignatura (Bloques I y II).

Las tareas tienen un enfoque práctico y se centran en desarrollar aplicaciones. La mayoría consisten en extender la funcionalidad de videojuegos creados por el propio estudiante mediante recursos y explicaciones aportadas por el profesor durante las clases.

En la tarea T1 –Programación de eventos I– los estudiantes debían ampliar una aplicación para dibujar con el dedo en tres colores (seleccionable mediante botones) sobre un *canvas* y poner una imagen en el fondo del *canvas* sobre la que pintar los trazos.

La tarea T2 –Programación de eventos II– consistió en incorporar al juego "Atrapa el topo"[4] la funcionalidad de reproducción de sonido y vibración del móvil cuando el topo es capturado.

La tarea T3 –Bloques condicionales– se centraba en extender el juego "Fruit loot"[5] de tal forma que al pulsar el botón de inicio del juego el *spicker* se debe colocar en la posición central del escenario del juego.

Por último, la tarea T4 –Bloques de repetición– consistía en realizar un desarrollo desde cero de un juego *Quiz* educativo (ver Figura 3). El jugador debe responder a varias preguntas mostradas por la aplicación obteniendo una puntuación según sus aciertos y errores. La aplicación muestra para cada pregunta varias opciones, siendo solo una de ellas correcta. El tema sobre el que trataba el *Quiz* era libre y lo escogía el/la estudiante. Sin embargo, se exigía que cumpliese las siguientes condiciones:

- En cada pregunta debe mostrar una imagen relacionada con el tema de la pregunta.
- Debe haber como mínimo 5 preguntas.
- Se mostrarán 4 opciones por cada pregunta y el jugador solo podrá seleccionar una.
- La aplicación no dará *feedback* sobre la corrección de la contestación seleccionada, pero sí llevará la cuenta de las preguntas contestadas correcta e incorrectamente.
- Un control *Label* muestra el resultado de la puntuación obtenida indicando el número de respuestas contestadas correcta o incorrectamente.

Los entregables de las tres primeas tareas y de la cuarta eran diferentes. Para las tres primeras tareas, era suficiente con entregar el fichero .APK ejecutable en Android (algunos estudiantes decidieron aportar además un documento breve con la explicación de la aplicación desarrollada y capturas del código fuente). Sin embargo, en la tarea T4 se les exigía entregar una memoria explicando el diseño de la interfaz de usuario que se había realizado y la programación de bloques

[4] https://appinventor.mit.edu/explore/ai2/molemash

[5] https://nostarch.com/programwithappinventor

realizada. Se indicó además que, si el programa era muy extenso, se podrían centrar las explicaciones en los principales bloques y utilizar anexos para aportar el programa completo. También se pedía entregar el proyecto de App Inventor (fichero .AIA) y el instalable Android (fichero .APK). Todo el material debía ir empaquetado en un fichero .ZIP o .RAR. La Figura 3 muestra algunas capturas de aplicaciones creadas por estudiantes de la tarea T4.

Figura 3: Aplicaciones de los estudiantes creadas en la tarea T4 (en la izquierda en un emulador y a la derecha en un *smartphone*).

En el enunciado de la tarea T4 se indicó el siguiente criterio de corrección para el programa de videojuego y la memoria creados por los estudiantes. Hay que notar que estos criterios son independientes de la dificultad de superar el juego programado por los estudiantes:

- Aprobado: el juego debe tener como mínimo 5 preguntas y se deben usar bloques *List* y el control *spinner*.
- Notable: el juego lleva la cuenta de las contestaciones correctas e incorrectas y al finalizar se informa de las mismas.
- Sobresaliente: se valora la calidad de la memoria (si contiene índice, estructura apropiada, claridad de las ideas, etc.), calidad de la interfaz de usuario (uso apropiado de controles, diseño, etc.) y calidad de la programación (programación correcta de los bloques, uso apropiado de variables, uso de procedimientos, etc.).

En el caso de las tres primeras tareas (T1-T3) se consideró innecesario incluir rúbricas o criterios de evaluación ya que las tareas eran muy simples y concretas.

5. Valoración de los alumnos

En esta sección se describen los resultados de una encuesta de opinión de los alumnos sobre variados aspectos de la asignatura.

5.1. Objetivo e instrumento

El objetivo de la encuesta era conocer la percepción que los estudiantes tienen sobre la dificultad y utilidad de los distintos lenguajes de bloques, así como sobre la utilidad de los recursos docentes y herramientas que se han usado en la asignatura. Como consecuencia, se esperaba identificar los puntos débiles del primer año de impartición, de forma que pudieran realizarse intervenciones docentes de mejora al siguiente curso.

Como instrumento de recogida de información, se diseñó un cuestionario organizado en varias partes:

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

En tres primeras partes del cuestionario, se han utilizado preguntas con una escala Likert que va desde 1 (“poco o nada”), pasando por una valoración intermedia de 3 (“aceptable”) hasta llegar a 5 (“mucho”). Para el sistema de evaluación, se ha utilizado una pregunta de dos opciones (sí/no) y una pregunta abierta. Por último, para recabar la opinión del estudiante se han formulado dos preguntas abiertas.

En las tres primeras partes del cuestionario se distinguió entre la primera y la segunda parte de la asignatura (Bloques I y II frente al Bloque III, respectivamente). Las dos últimas partes del cuestionario valoran aspectos globales de la asignatura.

La encuesta se difundió por medio del aula virtual una vez realizado el examen y publicadas las notas de la asignatura. Como paso previo a contestar online al cuestionario, se explicaba la voluntariedad y el objetivo de esta actividad y los alumnos debían marcar explícitamente su consentimiento.

5.2. Muestra y resultados

La participación en la evaluación fue voluntaria y los datos se trataron anónimamente. Se recogieron respuestas de 11 estudiantes de la asignatura.

En relación a los recursos docentes utilizados (parte 1 del cuestionario), la Figura 4 muestra los resultados

obtenidos. Se puede ver que los recursos más valorados (con valoración máxima de "mucho") son los vídeos explicativos del profesor y la realización de ejercicios prácticos. Lo que menos valoran y utilizan los estudiantes son las presentaciones tipo PowerPoint y las tutorías online con el profesor. También se aprecia que prácticamente no hay diferencia en el uso y valoración de estos tipos de recursos al tratarse de los lenguajes más simples como Scratch o de un lenguaje más complejo como App Inventor (Figura 4).

En relación con la utilidad del entorno de programación y la dificultad de aprender el lenguaje de programación (partes 2 y 3 del cuestionario), el Cuadro 1 muestra algunas medidas de estadística descriptiva. Para facilitar la presentación de los resultados, definimos variables que miden la utilidad percibida para cada uno de los lenguajes y otras que miden la dificultad percibida. El nombre de estas variables comienza con el prefijo "Util" o "Dif", expresando si mide utilidad o dificultad, seguido de las iniciales del lenguaje de programación. Por ejemplo, la variable Util_Cod mide el nivel de utilidad del lenguaje de Code.org.

Se puede ver que el entorno de programación más útil para aprender a programar (parte 2) corresponde a Scratch, seguido de ScratchJr, siendo Code.org el menos útil. En lo referente a su dificultad (parte 3), los lenguajes reciben puntuaciones acordes con la complejidad de los lenguajes, desde la mayor sencillez de Code.org hasta la mayor complejidad de App Inventor.

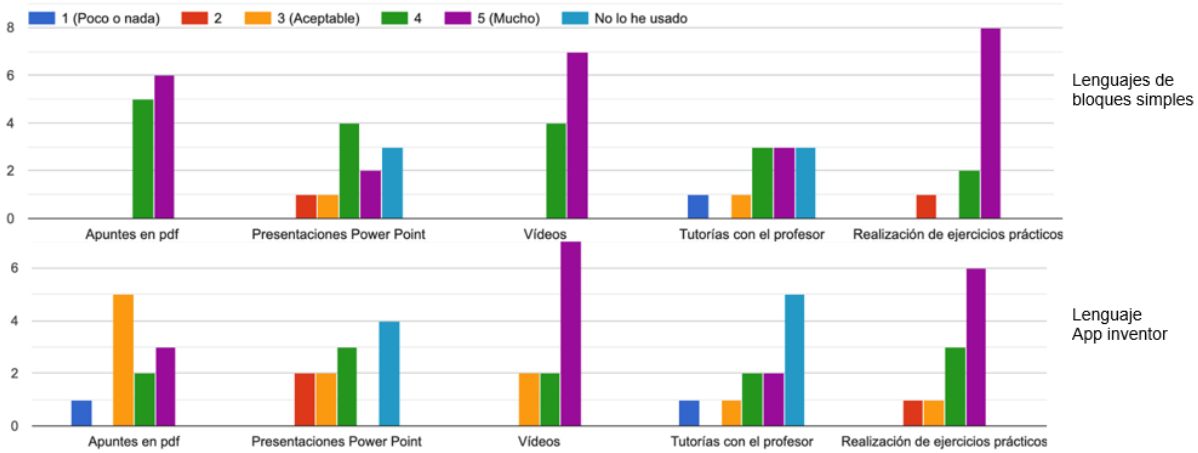

Figura 4: Utilidad de los recursos docentes según el tipo de lenguajes utilizados.

Variable (N=11)	Media	Mediana	Desviación estándar
Util_Cod	3,82	4	0,874
Util_ScJr	4,45	5	0,688
Util_Sc	4,73	5	0,467
Util_App	4,18	4	0,874
Dif_Cod	2,09	2	1,300
Dif_ScJr	2,27	2	1,191
Dif_Sc	2,91	3	1,221
Dif_App	3,73	4	1,421

Cuadro 1: Estadística descriptiva de las variables de percepción de utilidad y dificultad.

En cuanto al sistema de evaluación (parte 4 del cuestionario), 9 estudiantes (82%) lo consideran apropiado. A dos estudiantes que no les pareció apropiado, y consideraban que era innecesario el examen y que el caso práctico de desarrollo de App Inventor debía estar mejor alineado con los contenidos de la asignatura.

Finalmente, la parte 5 del cuestionario contenía dos preguntas abiertas. Una cuestión les preguntaba por las partes más difíciles de la asignatura, en caso de haber alguna. Se recogieron 7 respuestas, de las que 6 indicaban que era la parte de App Inventor, mientras que la otra respuesta aludía genéricamente a la actividad de programar: "*saber qué bloques combinar para cumplir con las instrucciones dadas*".

Una segunda pregunta pedía la opinión de los estudiantes sobre la asignatura, recogiéndose de todos los alumnos una o varias afirmaciones. Todos los alumnos expresaron su satisfacción con la asignatura, la planificación y la utilidad para su futura labor docente, como se puede apreciar en algunas frases extraídas literalmente de las encuestas: "*en general estoy bastante contenta con la asignatura y considero muy interesantes las herramientas utilizadas*", "*sin apenas conocimientos de programación se pueden hacer grandes recursos docentes*" y "*mi nivel en programación era nulo, considero que he aprendido mucho*". Algunos alumnos incluso expresaron su interés en aprender más de programación: "*(...) y estoy deseando ver qué hay en la segunda parte*" y "*lástima que no tenga más carga lectiva en comparación con otras asignaturas*".

Sin embargo, también se recogen algunas impresiones sobre aspectos negativos de la evaluación de la asignatura y de la dificultad del lenguaje App Inventor:

"*considero que introducir un sumatorio en el examen final ha sido algo complejo*" [en relación a la pregunta de App Inventor en la prueba de desarrollo], "*el salto de Scratch a App Inventor ha sido muy grande* [en dificultad]". Sobre este último aspecto, dos alumnos recuerdan que no tenían ningún conocimiento previo de programación, mientras que otro lamenta no haber podido profundizar más en App Inventor.

6. Debate

El resultado de la experiencia mostrada puede considerarse, en términos generales, satisfactorio pero con oportunidades de mejora. El propio diseño de la asignatura ha sido complejo porque se dispone de poco tiempo para cubrir un abanico de lenguajes basados en bloques, adecuados para ser utilizados en todas las etapas educativas preuniversitarias, desde Educación Infantil hasta Bachillerato e incluso la universidad. Esta restricción temporal impide profundizar en el conocimiento de ningún lenguaje lo suficiente como para poderse hablar de maestría en el mismo. Sin embargo, la introducción incremental de los lenguajes, unido a las características propias de los lenguajes basados en bloques, ha permitido que los alumnos hayan tenido éxito en el aprendizaje de la programación.

Los resultados peores sobre los lenguajes usados (véase Cuadro 1) se refieren a la utilidad del entorno de Code.org y a la dificultad de App Inventor. La razón de introducir Code.org fue proporcionar una forma muy simple de comenzar a programar, consistente en resolver "puzles" mediante programas sencillos [14]. Sin embargo, podría empezarse directamente con ScratchJr, que también es sencillo. En este caso, podría dedicarse una semana adicional a Scratch, dedicando más tiempo a los bloques más importantes para la transición a App Inventor, como son datos, control condicional y bloques de usuario. Asimismo, en el Bloque III debería mostrarse la correspondencia entre algunos conceptos de programación de Scratch y App Inventor. En otros estudios se ha identificado la importancia de percibir la utilidad de los lenguajes de programación y de cuidar la transición entre dos lenguajes [13].

Los bloques I y II se diseñaron a partir de tres decisiones didácticas principales: presentación incremental de tres lenguajes, descripción mediante apuntes del comportamiento de los bloques (llegando a identificar la información que constituye el estado de la ejecución de un programa) y tests con ejercicios orientados a la comprensión del lenguaje, más que al desarrollo. Un reto que queda abierto para el curso próximo es la posibilidad de reforzar o de introducir elementos de alguna otra didáctica existente para programación basada en bloques. Presentamos las posibilidades que nos parecen más prometedoras, aunque es evidente que no se pueden atender todas en las seis semanas dedicadas a ScratchJr y Scratch:

- Progresión a través de actividades previas al desarrollo de programas. Un ejemplo destacado para la programación basada en bloques es la didáctica "Use-Modify-Create" [11]. Los alumnos comienzan teniendo disponibles programas, que usan y modifican ligeramente para después realizar modificaciones más grandes. Actualmente, ya dejamos disponibles en el aula virtual algunos programas Scratch. Habría que valorar si estos programas son adecuados para modificar y ampliar o si es preferible construir otros nuevos.
- Apoyo en una máquina virtual educativa para explicar el efecto de ejecutar las instrucciones del lenguaje (máquinas nocionales [17]). Actualmente presentamos el comportamiento dinámico de los bloques e identificamos la información que constituye el estado de un programa [18]. Un paso más consistiría en modelar una máquina virtual que represente explícitamente el estado de ejecución de los programas.
- Ejercicios. En la medida de los posible, se han adaptado a la programación basada en bloques ejercicios que han mostrado su utilidad para mejorar la comprensión de la programación textual, como son los ejercicios predictivos [12]. En todo caso, conviene que sean ejercicios de corrección automática, para proporcionar realimentación inmediata al alumno, normalmente en formato de preguntas de respuesta múltiple [8].
- Patrones elementales [1]. Proporcionan soluciones estándar para acciones frecuentes. Facilitan la adquisición de un buen estilo de programación y la formación de esquemas mentales viables.

7. Conclusiones

Se ha presentado el diseño de una asignatura cuatrimestral para el aprendizaje de la programación basada en bloques y la experiencia del primer curso de impartición. Los alumnos son profesores en formación o en activo de todas las etapas educativas preuniversitarias. También se han presentado los resultados de un cuestionario, creado para recoger la percepción de los alumnos sobre el curso. Como consecuencia, se han identificado algunos puntos fuertes y débiles y se han esbozados posibles soluciones para estos últimos, que habrá que analizar con detenimiento para el próximo curso académico. La asignatura es parte de un esfuerzo de nuestro grupo de investigación sobre la didáctica de los lenguajes basados en bloques, que puede resultar útil a profesores de asignaturas similares.

Referencias

[1] Kashif Amanullah y Tim Bell. Teaching resources for young programmers: The use of patterns. En *Proc. 2020 IEEE Global Engineering*

Education Conf., EDUCON, Alberto Cardoso, Gustavo R. Alves y Teresa Restivo (eds.), págs. 679-687, 2020. DOI: 10.1109/EDUCON45650.2020.9125179.

[2] L.W. Anderson, D.R. Krathwohl, P.W. Airasian, K.A. Cruikshank, R.E. Mayer, P.R. Pintrich, R. Raths y M.C. Wittrock. *A Taxonomy for Learning, Teaching and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*. Pearson Education Limited. 2001.

[3] Michal Armoni y Moti Ben-Ari. *Computer Science Concepts in Scratch*, Weizmann Institute of Science. 2013. Disponible en https://stwww1.weizmann.ac.il/scratch/scratch_en/.

[4] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon y Franklyn Turbak. Learnable programming: Blocks and beyond. *Communications of the ACM*, 60(6):72-80, junio 2017. DOI: 10.1145/3015455.

[5] Karen Brennan, Christan Balch y Michelle Chung. *Informática Creativa*. Harvard Graduate School of Education. Disponible en http://scratched.gse.harvard.edu/guide/curriculum.html.

[6] 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.

[7] L. P. Flannery, E. R. Kazakoff, P. Bontá, B. Silverman, M. U. Bers y M. 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. DOI: 10.1145/2485760.2485785.

[8] Pedro Henriques Abreu, Daniel Castro Silva y Anabela Gomes. Multiple-choice questions in programming courses: Can we use them and are students motivated by them? *ACM Trans. Computing Education*, 19(14):artículo 6, noviembre 2018. DOI: 10.1145/3243137.

[9] INTEF. *Marco Común de Competencia Digital Docente*, septiembre 2017.

[10] Joint Informatics Europe & ACM Europe Working Group on Informatics Education. *Informatics education: Europe cannot afford to miss the boat*. 2013. Disponible en: http://europe.acm.org/iereport/ACMandIEreport.pdf.

[11] I. Lee, F. Martin, J. Denner, B. Coulter, W. Allan, J. Erickson, J. Malyn-Smith y L. Werner. Computational thinking for youth in practice. *Inroads*, 2(1):32-37, marzo 2011. DOI: 10.1145/1929887.1929902.

[12] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon y L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119-150, diciembre 2004.

[13] José Alfredo Martínez Valdés, J. Ángel Velázquez Iturbide y Raquel Hijón Neira. A (relatively) unsatisfactory experience of use of Scratch in CS1. En *Proc. 5th International Conf. Technological Ecosystems for Enhancing Multiculturality, TEEM'07*. 2017. DOI: 10.1145/3144826.3145356.

[14] Radek Pelánek y Tomáš Effenberger, Design and analysis of microworlds and puzzles for block-based programming. *Computer Science Education*. 2020. DOI: 10.1080/08993408.2020.1832813.

[15] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman e Y. Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, noviembre 2009. DOI: 10.1145/1592761.1592779.

[16] 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/.

[17] Joha Sorva. Notional machines and introductory programming education. *ACM Trans. Computing Education*, 13(2):artículo 8. 2013. DOI: 10.1145/2483710.2483713.

[18] J. Ángel Velázquez Iturbide. Towards the design of notional machines for simple block-based languages. En *SIIE 2021, 2021 International Symposium on Computers in Education*, Antonio Balderas, Antonio Mendes y Juan Manuel Dodero (eds.), IEEE Xplore, 2021. DOI: 10.1109/SIIE53363.2021.9583645.

[19] J. Ángel Velázquez Iturbide y Mercedes Martín Lope. Análisis del "pensamiento computacional": una perspectiva educativa. *RED. Revista de Educación a Distancia*, 21(68), artículo 6, noviembre 2021. DOI: 10.6018/red.484811.

[20] David Weintrop, Block-based programming in computer science education. *Communications of the ACM*, 62(8):22-25, agosto 2019. DOI: 10.1145/3341221.

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

[22] A. Yadav, N. Zhou, C. Mayfield, S. Hambrusch y J. T. Korb. Introducing computational thinking in education courses. En *Proc. 42nd ACM Technical Symp. Computer Science Education, SIGCSE'11*, pp. 465-470, 2011. DOI: 10.1145/1953163.1953297.