El desarrollo del pensamiento computacional como formación básica en carreras de Ingeniería

Patricia Calvo[1] y Arturo Servetto[1]

[1] Universidad de Buenos Aires. Facultad de Ingeniería. Departamento de Computación. Buenos Aires, Argentina.
{pmcalvo, aservetto}@fi.uba.ar

Resumen. Se presenta un desarrollo metodológico para la enseñanza y el aprendizaje de Algoritmia y Programación empleando el Entorno Integrado de Desarrollo y Aprendizaje del lenguaje de programación Python, en cuatro cursos de una asignatura de formación básica de carreras de Ingeniería; los cursos se desarrollaron en modalidad virtual en el primer cuatrimestre del año 2021, para los que se presentan resultados, y se continúan desarrollando actualmente en modalidad semipresencial. La asignatura tiene un diseño curricular basado en competencias, y la metodología que se comparte se basa en actividades grupales colaborativas focalizadas en unidades temáticas para desarrollar competencias específicas y genéricas en un marco de evaluación formativa continua, y en la promoción del cursado por integración de contenidos en un proyecto colaborativo con evaluación sumativa.

Palabras clave: Desarrollo Metodológico, Algoritmia y Programación, Enseñanza de Programación en Carreras Tecnológicas, Desarrollo de Competencias, Actividades Formativas.

The development of computational thinking as basic training in engineering careers

Abstract. A methodological development is presented for the teaching and learning of Algorithm and Programming using the Integrated Development and Learning Environment of the Python programming language, in four courses of a basic training subject of Engineering careers; the courses were developed in virtual mode in the first four-month period of the year 2021, for which results are presented, and are currently being developed in blended mode. The course has a competency-based curriculum design, and the shared methodology is based on collaborative group activities focused on thematic units to develop specific and generic competencies within a framework of continuous formative evaluation, and on the promotion of the course by integration of contents in a collaborative project with summative evaluation.

Keywords: Methodological Development, Algorithm and Programming, Teaching of Programming in Technological Careers, Competence Development, Training Activities.

1 Introducción

1.1 Problemática pedagógica

El aprendizaje de los contenidos de la asignatura Computación como ciencia básica en la Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA) se focaliza en contribuir a desarrollar en los estudiantes de las carreras de Ingeniería en general la capacidad de construir la solución de problemas con la computadora (mediante el ingenio de algoritmos y su codificación como programas). El mayor desafío cognitivo al que se enfrentan los alumnos es el descubrimiento de un algoritmo efectivo (eficaz y eficiente) que solucione el problema planteado [1].

En general, los alumnos de Computación son estudiantes que recién ingresan a la FIUBA en un contexto curricular de alta concentración de asignaturas de ciencias básicas que cursan en simultáneo. La demanda creciente de tiempo y esfuerzo hace que frecuentemente prioricen a otras asignaturas que, por el sistema de correlatividades, si no son aprobadas ralentizan el avance en sus respectivas carreras, por lo cual se verifica una alta deserción en las primeras semanas de cursada de Computación. Por otra parte, al comenzar el cursado se suele observar en los alumnos escasa motivación o interés por los contenidos de la asignatura por no percibir la afinidad con sus campos profesionales [2].

El pensamiento computacional es un paradigma para resolver problemas que en muchas ocasiones a los nuevos estudiantes les resulta algo extraño; éstos a menudo evidencian serias dificultades para elaborar abstracciones, dan preponderancia a los hábitos de aprendizaje mecánico y memorístico, y manifiestan dificultades tanto para analizar problemas como para articular estrategias de resolución [3].

Las competencias de la asignatura que los autores se proponen que adquieran los alumnos no sólo son cognitivas (saber), procedimentales/instrumentales (saber hacer), y actitudinales (ser), sino también transversales o genéricas (asociadas a actividades, no a contenidos) [4]:

Competencias cognitivas. Conocimiento de vocabulario técnico que facilite la comunicación con profesionales de la Informática para el trabajo interdisciplinar. Conocimiento general sobre algoritmia como paradigma de resolución de problemas. Conocimientos básicos de la sintaxis de un lenguaje de programación de alto nivel. Conocimiento de estructuras de datos fundamentales. Conocimientos básicos de complejidad computacional.

Competencias procedimentales/instrumentales. Capacidad para descomponer un problema real para su posterior codificación en un programa; documentar programas con claridad y sencillez; analizar la complejidad de algoritmos; interpretar y utilizar las diferentes estructuras de control y de datos para implementar soluciones a problemas específicos; comprender documentación técnica y reutilizar código desarrollado por terceras partes.

Competencias actitudinales. Capacidad de resolución de problemas mediante algoritmos. Motivación por la claridad, sencillez y eficiencia en la resolución de problemas. Capacidad para debatir y concluir las distintas soluciones a un problema.

Competencias transversales o genéricas. Capacidad para la autoorganización y planificación del trabajo individual y del proceso de aprendizaje. Capacidad para el trabajo en grupo. Capacidad de análisis y síntesis. Motivación por la calidad del resultado. Disposición para el compromiso, la responsabilidad, la colaboración, el empeño, la dedicación, la solidaridad, la honestidad y el respeto en el trabajo individual y grupal durante el proceso de construcción del conocimiento.

1.2 Fundamentos metodológicos

Para que los estudiantes sean capaces de ingeniar algoritmos y comunicarlos a la computadora, el profesor debe enseñarles a pensar computacionalmente en el proceso de construcción de una solución lógica.

Sostiene Niklaus Wirth que nuestra herramienta mental más importante para competir con la complejidad es la abstracción, y que, por tanto, un problema no deberá considerarse inmediatamente en términos de instrucciones de un lenguaje, sino de elementos naturales del problema mismo, abstraídos de alguna manera.

Para solucionar problemas con la computadora los estudiantes utilizan el Modelo de Solución de Problemas de Pólya [5] aplicado al ámbito de la construcción de programas que consta de cuatro fases:

1. **Análisis.** El estudiante debe comprender en qué consiste el problema a resolver con la construcción del programa.
2. **Diseño.** El estudiante debe diseñar una estrategia para definir los recursos y descubrir el algoritmo.
3. **Codificación.** El estudiante debe implementar la estrategia para construir el programa.
4. **Evaluación.** El estudiante debe ejecutar el programa construido para comprobar que la solución es correcta. El Análisis del problema y el Diseño de la solución, son las fases algorítmicas y constituyen el desafío creativo de aprendizaje porque requiere que los estudiantes desplieguen el pensamiento computacional. La Codificación y la Evaluación son las fases de programación. La Algoritmia y la Programación no se aíslan, sino que se distinguen y se integran [6].

El pensamiento computacional es una forma específica de pensar que propicia el análisis y la relación de ideas para la organización y la representación lógica de procedimientos [7]. Según Wing [8], de manera informal, el pensamiento computacional describe la actividad mental al formular un problema para admitir una solución computacional, se superpone con el pensamiento lógico y el pensamiento sistémico, incluye el pensamiento algorítmico y el pensamiento paralelo, que a su vez involucran otros tipos de procesos de pensamiento. Simari [9] explicita que Wing describe esta forma de pensamiento como una combinación de las formas de pensamiento matemático e ingenieril.

2 Descripción de la metodología didáctica

2.1 Dispositivos y estrategias

Se diseñaron estrategias pedagógicas en la virtualidad para promover el desarrollo del pensamiento computacional como competencia clave, articular con otras formas de pensamiento (lógico, matemático, ingenieril) y favorecer procesos de análisis, síntesis, creatividad y comunicación en la formación de estudiantes de ingenierías [10].

Las estrategias de enseñanza y de aprendizaje de los contenidos de Computación se sintetizan en la experiencia de pensar para crear juntos. Integran enseñar a pensar y enseñar contenidos con el propósito de que los estudiantes aprendan a pensar computacionalmente.

El desarrollo de los contenidos teóricos y prácticos se aborda desde tres núcleos centrales (algoritmo, programa y computadora) en forma interrelacionada, iterativa e incremental y se organiza en espiral. Durante las primeras 10 semanas de clases el descubrimiento de algoritmos y el desarrollo de programas se complejiza con la incorporación gradual de nuevas herramientas de programación y el conocimiento de la computadora se profundiza.

La enseñanza, el aprendizaje y la evaluación de los contenidos de Computación constituyen una triada compleja que requiere a los estudiantes de las carreras de Ingeniería el desarrollo del pensamiento computacional a través de la Algoritmia y la Programación como un paradigma de resolución de problemas que contribuye a su formación profesional.

Se diseñaron dispositivos didácticos como modelos pedagógicos utilizados tanto por los profesores, para enseñar a programar y evaluar las soluciones de los estudiantes, como por los estudiantes, para aprender a programar y autoevaluar la calidad de sus producciones:

- **Modelo de Programa Tipo.** Archivo creado con el editor del entorno integrado de desarrollo y aprendizaje de Python (programa.py), para estructurar el diseño y la codificación de un algoritmo;

- **Tabla de Preguntas Formativas.** Rúbrica interrogativa para establecer y aplicar criterios de diseño y de evaluación de programas.

Ambos dispositivos constituyen un andamiaje para organizar la solución de un problema en un procedimiento algorítmico a partir del enunciado del problema, hacer visible el pensamiento computacional y expresar el algoritmo en lenguaje Python. Y contribuyen al aseguramiento del logro de una solución efectiva (eficaz y eficiente) al problema propuesto.

Se diseñaron actividades formativas a distancia que comprenden los temas asociados a los contenidos prácticos y posibilitan el avance en el conocimiento de las herramientas de programación. Cada semana (1 a 10) los estudiantes desarrollan programas con sus pares (grupos de 2 o 3 integrantes) y los entregan como tareas en el aula virtual (cursos virtuales gestionados con Moodle).

Las actividades tienen calificación conceptual y devolución formativa porque la corrección es detallada y se focaliza en los criterios de funcionalidad, metodología y calidad de diseño para el logro de programas efectivos. El objetivo de la devolución formativa es que los estudiantes se autoevalúen y autorregulen su aprendizaje. Se publica en el aula virtual con el propósito de que cada participante pueda visibilizar las observaciones al grupo propio y aprender de las correcciones comunicadas a los demás grupos.

Se diseñó un proyecto o trabajo final integrador a distancia (juego *Batalla Naval* o *Hundir la Flota*) que comprende todos los temas asociados a los contenidos prácticos aprendidos a través del desarrollo de las actividades formativas grupales. En las últimas 4 semanas de clases (13 a 16) los estudiantes desarrollan programas con sus pares (grupos de 2 o 3 integrantes) por etapas (1 a 4) y cargan los archivos como tareas en el aula virtual.

En cada etapa se debe entregar una versión del programa o prototipo incremental que cumpla los objetivos y utilice los recursos especificados (en cada versión se deberá desarrollar una o más funciones y reemplazar el código del programa principal), más el enlace a una grabación en Google Meet de aproximadamente 20 minutos en la que, interviniendo todos los integrantes de cada grupo, cuenten cómo pensaron la solución, de manera informal (no leyendo lo que escribieron), pero presentando en pantalla la plantilla (emulando la forma en que trabajamos en cada clase para desarrollar los programas): "para resolver esta etapa se nos ocurrió... y entonces, en análisis de casos... nos dimos cuenta que íbamos a necesitar tales recursos... y luego, en la síntesis de casos, organizamos la secuencia de acciones... y para codificar el algoritmo en el Prólogo... y luego en la Resolución... ". Es decir, una síntesis de la algoritmia y de la programación.

En una clase/encuentro posterior a una entrega, se puede pedir a los miembros presentes de un grupo que amplíen o aclaren algún aspecto de la versión entregada.

Cada integrante de un grupo recibe una devolución formativa de los programas desarrollados en cada etapa por mensajería interna del curso virtual. La calificación del trabajo final integrador es cuantitativa (4 a 10) y el proyecto debe ser aprobado para aprobar la cursada de la asignatura.

Las enunciaciones y entregas tanto de las actividades formativas como del trabajo final integrador por etapas se instrumentaron mediante tareas de Moodle, y se establecieron pautas de documentación y condiciones de trabajo y de entrega verificables para fomentar competencias no sólo procedimentales sino también actitudinales. Los alumnos tuvieron la posibilidad de realizar consultas sobre las actividades asincrónicas mediante foros o mensajería interna del curso virtual, así como también en los encuentros sincrónicos.

2.2 Desarrollo

La experiencia pedagógica se desarrolló en cuatro cursos de Computación (cuatro horas de clases por semana), cada uno a cargo de un Profesor Responsable y con un promedio de 40 alumnos por curso.

Mediante conexiones sincrónicas vía Google Meet, las clases a distancia se enfocaron en el desarrollo de algoritmos y programas con la participación de los estudiantes para solucionar casos de estudio. El profesor utiliza el Modelo de Programa Tipo como estructura guía para modelar el pensamiento computacional en la búsqueda de una solución viable, y la Tabla de Preguntas Formativas para el aseguramiento de un buen diseño del algoritmo. Se busca favorecer la colaboración entre pares y profesor para que los estudiantes logren desarrollar las capacidades de análisis (descomposición del problema en subproblemas), síntesis (composición de subproblemas para obtener la solución final), y evaluación (búsqueda de eficiencia). Y luego, de codificar el algoritmo.

Los encuentros sincrónicos se grabaron y pusieron a disposición de los estudiantes en el aula virtual de cada curso (Campus Virtual FIUBA en Moodle). Como material de estudio y de trabajo se cargaron también archivos de los programas desarrollados en clase y programas ejemplo. Como devolución general, luego de la entrega grupal de las soluciones a las actividades formativas, se publicó una solución posible a cada problema planteado para que los integrantes de los diferentes grupos pudieran autoevaluar la calidad de la producción propuesta antes de recibir la devolución grupal personalizada. También se utilizó el aula virtual para la comunicación asincrónica a través de foros (anuncio de novedades por parte del profesor, atención de consultas tanto individuales como grupales relacionadas con la comprensión de los contenidos de la asignatura y/o de la organización del curso).

Las consultas y devoluciones personalizadas, individuales y grupales, asincrónicas, fueron realizadas a través del correo electrónico de la asignatura, y las sincrónicas, mediante reuniones a través de la plataforma Google Meet.

Tanto las actividades formativas como el trabajo final integrador fueron grupales colaborativos. Esto implicó que el desarrollo de los algoritmos debía realizarse de forma consensuada por todo el grupo, lo cual requiere aplicar habilidades de negociación y capacidad para exponer el punto de vista propio frente a los demás integrantes, siendo todos los miembros corresponsables del producto final. Para estas elaboraciones los grupos contaron permanentemente con el apoyo y orientación de los docentes a través de las diferentes vías de comunicación establecidas.

En la Tabla 1 se presenta un ejemplo de uso de la plantilla de programas tipo, y en la Tabla 2 se detallan las preguntas guía para la evaluación formativa.

Tabla 1. Programa en Python para introducción de estructuras de selección múltiple.

romanos.py

```
'''
Objetivo del programa (descripción del problema que resuelve): hallar la representación romana de un número entero entre 1 y 3999.

Autor/es: 2021C1

Versión: 1

Fecha: 04/2021

Análisis de Casos:
1. 240 >> CCXL: 240//1000=0 (''); (240//100)%10=2 (CC);  (240//10)%10=4 (XL); 240%10=0 ('')
2. 409 >> CDIX: 409//1000=0 (''); (409//100)%10=4 (CD);  (409//10)%10=0 (''); 409%10=9 (IX)
3. 1034 >> MXXXIV: 1034//1000=1 (M); (1034//100)%10=0(''); (1034//10)%10=3 (XXX); 1034%10=4 (IV)
4. 2678 >> MMDCLXXVIII: 2678//1000=2 (MM); (2678//100)%10=6 (DC); (2678//10)%10=7(LXX); 2678%10=8 (VIII)
5. 3999 >> MMMCMXCIX: 3999//1000=3 (MMM); (3999//100)%10=9 (CM); (3999//10)%10=9 (XC); 3999%10=9 (IX)
6. 1 >> I: 1//1000=0 (''); (1//100)%10=0 (''); (1//10)%10=0 (''); 1%10=1 (I)
7. 10 >> X: 10//1000=0 (''); (10//100)%10=0 (');   (10//10)%10=1(X); 10%10=0 ('')
8. 100 >> C: 100//1000=0 (''); (100//100)%10=1 (C);  (100//10)%10=0 (''); 100%10=0 ('')
9. 1000 >> M= 1000//1000=1 (M); (1000//100)%10=0 ('); (1000//10)%10=0 (''); 1000%10=0 ('')

Síntesis de Casos (composición de casos para la generalización):
mr: nro//1000 << Transformación de cifra decimal nro//1000 obtenida a su expresión romana para la unidad de mil según
su orden de magnitud
cr: (nro//100)%10 << Transformación de cifra decimal (nro//100)%10 obtenida a su expresión romana para la centena según
su orden de magnitud
dr: (nro//100)%10 << Transformación de cifra decimal (nro//10)%10 obtenida a su expresión romana para la decena según su
orden de magnitud
ur: nro%10 << Transformación de cifra decimal nro%10 obtenida a su expresión romana para la unidad según su orden de
magnitud
```

```
romano: mr + cr + dr + ur << Representación romana del número decimal

Recursos (variables y funciones del programa -nombre, propósito, y si se transfor-
man o no en el programa)

   Datos que solicitar al usuario (sea en el prólogo o sea durante la resolución):
   nro: entero positivo entre 1 y 3999, no se transforma en el programa

   Auxiliares (necesarios para transformaciones intermedias):
   u, d, c, m: unidad, decena, centena y unidad de mil, dígitos enteros del número
               original, no se transforman en el programa
   ur, dr, cr, mr: cadenas (strings) para la conversión de cada dígito del número
                   original a su representación romana, no se transforman en el
                   programa

   Resultados (a informar sea durante el desarrollo o en el epílogo):
   romano: cadena con la representación del número original como romano, no se
           transforma en el programa

   Funciones propias (necesarias para la descomposición del problema en
   subproblemas):

'''
# DEFINICIÓN DE FUNCIONES (si se requieren para descomponer la solución del
# problema)

#1 PRÓLOGO
#1.1 Presentación
#1.1.1 Impresión del título del programa en pantalla
print('NÚMEROS ROMANOS')

#1.1.2 Descripción o aclaraciones al usuario (opcional)
print('\nRepresentación de números enteros entre 1 y 3999 en notación romana.')

#1.2 Datos iniciales
#1.2.1 Solicitud e ingreso de datos desde teclado (opcional, si los datos se piden
# durante la resolución)
nro=int(input('\nIngrese el número entre 1 y 3999 para obtener su representación
como romano: '))

#1.2.2 Establecimiento de valores iniciales para datos auxiliares o que se
# transformarán en resultados (opcional)
# Dígitos del número:
u = nro % 10
d = (nro // 10) % 10
c = (nro // 100) % 10
m = nro // 1000
```

```
#2 RESOLUCIÓN
# Descomposición del problema en subproblemas 2.x que a su vez pueden requerir
# ingreso o inicialización de datos o mostrar resultados

#2.2 Unidad
if u < 4: ur = u*'I' # las opciones deben ordenarse por frecuencia o posibilidad
# descendente para mejorar la eficiencia del programa
elif 4 < u < 9: ur = 'V' + (u-5)*'I'
elif u == 4: ur = 'IV'
else: ur = 'IX' # u==9

#2.3 Decena
if d < 4: dr = d*'X'
elif 4 < d < 9: dr = 'L' + (d-5)*'X'
elif d == 4: dr = 'XL'
else: dr = 'XC'

#2.4 Centena
if c < 4: cr = c*'C'
elif 4 < c < 9: cr = 'D' + (c-5)*'C'
elif c == 4: cr = 'CD'
else: cr = 'CM'

#2.5 Unidad de mil
mr = m*'M'

#2.6 Número romano
romano = mr + cr + dr + ur

#3 EPÍLOGO
#3.1 Muestra de la solución del problema por pantalla (opcional, si sólo se muestran
# resultados durante la resolución)
print(f'\nLa representación romana de {nro} es {romano}')

#3.2 Pausa para ver resultados en pantalla (que se puede obviar, si los resultados se
# van mostrando durante la resolución)
print() # salto de línea
input('Pulse Intro (Enter) para terminar el programa...') # pausa forzada
```

Tabla 2. Preguntas Guía para la Evaluación Formativa.

Aspecto	**Preguntas**
Funcionamiento	¿Comprende cuál es el problema que se debe resolver con el

del programa (programa eficaz)	programa? ¿El programa es eficaz? ¿Informa resultados correctos esperados? ¿Cumple con los requerimientos del enunciado? ¿El programa establece una comunicación efectiva con el usuario? ¿Solicita datos y exhibe resultados de manera clara, precisa, no ambigua? ¿Posibilita al usuario la ejecución del programa? ¿Solicita datos al usuario haciendo foco en el problema (no en el programa)? ¿Funciona correctamente el programa para todos los casos de prueba?
Metodología de desarrollo (programa inteligible)	¿Utiliza el Modelo de Programa Tipo (programa.py) para ingeniar y codificar el algoritmo? ¿Comprende cómo completar las secciones del encabezado? ¿Tiene en cuenta la sistematización del proceso algorítmico? ¿Realiza análisis de casos? ¿Trabaja con valores de los datos y de los resultados? ¿Considera todos los escenarios posibles? ¿Describe la transformación de datos en resultados? ¿Realiza síntesis de casos? ¿Trabaja con nombres de los recursos? ¿Encuentra un procedimiento general? ¿Especifica nombres y propósitos de todos los recursos y si se transforman o no en el programa? ¿Elige nombres significativos de recursos? ¿Describe los subproblemas en la Resolución? ¿Es inteligible la forma en que resuelve el problema con el programa? ¿Separa bloques de código en el texto del programa mediante una línea en blanco o un comentario de encabezamiento? ¿Completa de manera correcta las secciones del Prólogo, la Resolución y el Epílogo?
Calidad de diseño (programa eficiente)	¿Descubre la naturaleza del problema/subproblema a resolver? ¿Implementa la solución del problema/subproblema con la sentencia o estructura de datos apropiada según el avance del curso? ¿El programa es eficiente? ¿Se definen solamente recursos necesarios? ¿Se resuelve con las instrucciones necesarias? ¿Se evitan las instrucciones redundantes?

3 Resultados

En la Tabla 3 se presentan las respuestas a algunas preguntas realizadas a 34 alumnos (encuestados en 2 cursos) que aprobaron el cursado de la asignatura, como mues-

tra de la incidencia de la aplicación de la estrategia de enseñanza en la autoevaluación de los aprendizajes de los estudiantes.

Tabla 3. Encuesta a alumnos.

Preguntas	**Respuestas**
¿En general, con qué grado de intensidad participaste en la realización de las actividades grupales formativas?	Mucho 21 Bastante 11 De manera equitativa 1 Hice el 90% de todos los trabajos 1
¿En general, con qué grado de intensidad participaste en el desarrollo del proyecto de algoritmia y programación?	Mucho 20 Bastante 13 Siempre que pude a pesar de los problemas que tuvimos para coincidir todos 1
¿En tu grupo utilizaron la plantilla modelo de programa tipo programa-f.py como soporte para idear una estrategia de solución de cada problema propuesto?	Siempre 10 Muchas veces 8 Bastantes veces 5 Pocas veces 9 Para comentar el código 1 Primero creábamos el código y luego lo copiábamos a la plantilla modelo 1
¿Participaste en la fase de algoritmia (completar secciones objetivo, análisis, síntesis, recursos) para solucionar cada problema propuesto?	Siempre 18 Muchas veces 8 Bastantes veces 6 Pocas veces 2
¿Participaste en la fase de programación (completar secciones prólogo, resolución, epílogo) para solucionar cada problema propuesto?	Siempre 14 Muchas veces 15 Bastantes veces 5
¿En general, cómo trabajaron en tu grupo?	Por colaboración (todos los integrantes trabajamos en forma conjunta para resolver los problemas) 25 Por revisión (un integrante resolvió gran parte de los problemas y los demás revisamos las soluciones) 3 Otras respuestas (por cooperación y colaboración, comparación de soluciones individuales y elección de la mejor solución, etc.) 6
¿Cuánto ayudó (o no) trabajar con la plantilla modelo de programa tipo programa.py a hacer visible el pensamiento computacional en la solución de los problemas propuestos?	Mucho 11 Bastante 13 Poco 8 Otro (inteligibilidad) 2
¿Cuánto ayudó (o no) trabajar con la plantilla modelo de programa tipo programa-f.py para codificar el algoritmo en	Mucho 10 Bastante 15 Poco 9

lenguaje Python?	
¿Revisaste las devoluciones detalladas (aspectos funcionamiento del programa, metodología de desarrollo y calidad de diseño) de las entregas de las actividades formativas y las tuviste en cuenta para autoevaluar la solución propia y mejorar futuras entregas?	Siempre 13 Muchas veces 11 Bastantes veces 9 Pocas veces 1
¿Cuánto ayudaron las devoluciones en relación con el funcionamiento del programa en cada etapa de la entrega del proyecto de algoritmia y programación para tomar conciencia sobre los errores, reflexionar sobre la solución presentada y autorregular el propio aprendizaje?	Mucho 20 Bastante 12 Poco 2

Se presentan también algunas de las respuestas significativas a una pregunta realizada en la misma encuesta: ¿Cómo describirías tu propio proceso de aprendizaje para solucionar problemas con la computadora (algoritmos y programas) en este curso?

Proceso Muy Bueno (15 respuestas): *"Mi proceso de aprendizaje en este curso lo definiría como óptimo. Creo haber aprovechado las herramientas que el curso brindó y haber adquirido conocimientos sumamente útiles que ya ansío poner en práctica. Ya estoy trabajando en un programa que me encargaron en el que pongo en uso lo que he aprendido este cuatrimestre"; "aprendí mucho y de forma práctica"; "aprendí mucho más de lo que esperaba y por ende fue un proceso fructífero"; "aprendí mucho y me gustó"; "aprendí mucho no solo para la materia sino a pensar en resolver problemas de cualquier tipo"; "intenso"; "exponencial".*

Proceso Bueno (12 respuestas): *"Mi capacidad para resolver problemas arrancó siendo nula y hoy en escala de 1 a 10 es 7/10"; "me hizo caer en cuenta de malas prácticas que he tenido para programar"; "bastante bien, ya que arranqué de no saber nada a hacer una batalla naval".*

Proceso Dificultoso (7 respuestas): *"Siento que desde la primera actividad hasta la última progresé mucho y me dí cuenta que trabajo a trabajo me costaba menos pensar en los algoritmos que me llevarían a cumplir el objetivo propuesto. Además, yo venía frustrada de intentar aprender a programar con C, y con Python no me pasó lo mismo. Le sentí utilidad hasta para lo cotidiano desde la clase 1 y creo que es lo que me motivó y me ayudó a avanzar"; "Realmente mejoró mucho. Lo que quiero decir es que cuando inició la cursada tenía una cierta idea de cómo crear un algoritmo, pero no sabía aplicarla correctamente. Hoy, una vez finalizada la cursada, puedo sentir que estoy más preparado y con conceptos más sólidos a la hora de tener que escribir una solución para un problema".*

En cuanto a la modalidad de trabajo colaborativo, en uno de los cursos se realizó una encuesta a alumnos que aprobaron la asignatura para recabar datos sobre los con-

flictos que eventualmente surgieron durante el desarrollo de los trabajos con esa modalidad.

Entre la información obtenida se destaca la siguiente:

- Algunos alumnos tuvieron que resolver problemas de conectividad durante las reuniones grupales para desarrollar los trabajos.
- Algunos grupos mencionaron problemas de organización temporal debido a horarios de cursadas y/o de trabajos.
- Algunos grupos relataron que un problema a resolver fue la poca disponibilidad horaria de algunos integrantes, por lo cual se conversó para negociar una solución que permitiera la participación de todos.
- Frecuentemente tuvieron que negociar para llegar a un consenso. Esta problemática se resolvió de diferentes formas, entre las cuales se pueden mencionar las siguientes:
    - Se enunciaban distintas soluciones y se realizaba una votación para elegir la que parecía más conveniente.
    - Se debatía hasta que los integrantes quedaban convencidos de cuál era la mejor solución o estrategia de resolución, por unanimidad.
    - Se analizaban las distintas opciones de resolución aplicando los conceptos básicos de complejidad temporal adquiridos durante la cursada.

4 Conclusiones

Se diseñó la estrategia de enseñanza y de aprendizaje de los contenidos de Computación con el propósito de que los estudiantes adquieran la capacidad de pensar computacionalmente para crear juntos soluciones efectivas con algoritmos y programas a los problemas propuestos.

Los resultados de las encuestas evidencian en la mayoría de los estudiantes buenos procesos de aprendizaje durante el cursado de la asignatura que se corresponden con el rendimiento obtenido en las evaluaciones sumativas. Y, a su vez, motivan a profundizar y mejorar la utilización del Modelo de Programa Tipo y la Tabla de Preguntas Formativas en experiencias futuras como dispositivos didácticos clave para facilitar el descubrimiento de algoritmos, como así también, a enfatizar la práctica educativa a través de la evaluación continua (actividades y devolución formativas) y el trabajo colaborativo.

Los comentarios compartidos por algunos estudiantes en las encuestas impulsan a seguir trabajando en esta línea de acción hacia una educación en tecnología para la solución de problemas con la computadora: “Fue muy gratificante esta experiencia y muy lindo la participación en equipo y aprender programación.”; “El haber trabajado en grupo me sirvió muchísimo no solo para aprender sino para relacionarme interper-

sonalmente con gente de la facultad."; "Considero que la materia fue de gran utilidad para empezar a adquirir conocimientos del área de programación."; "Las actividades formativas fueron entretenidas y el proyecto final más todavía."; "En todo momento sentí que los contenidos que estábamos dando podían ser aplicables y útiles (cosa que no me pasa con otras materias)."

5 Agradecimiento

Los autores agradecen la colaboración de la Ing. Elizabeth Jiménez Rey, docente investigadora en la FIUBA hasta el 31 de agosto de 2021, para la elaboración de este artículo. Participó en el desarrollo de la experiencia educativa en dos de los cursos de la asignatura Computación durante el primer cuatrimestre de 2021 como parte del proyecto UBACyT 20020190100165 Ecosistemas tecnopedagógicos de aprendizaje activo en carreras de Ingeniería.

Referencias

1. Jiménez Rey, E., Aveleyra, E., Barranquero, F.: Competencias en Algoritmia y Programación como Formación Básica en Ingenierías: el Rol del Pensamiento Visible y la Mediación Tecnológica. IV CIECIBA Congreso Internacional de Enseñanza de las Ciencias Básicas, Actas de Resúmenes (2019) 14-15.
   Recuperado de: http://cieciba.multisitio.interior.edu.uy/presentacion-de-trabajos/
2. Jiménez Rey, E., Servetto, A., y Calvo, P.: Desarrollo de Pensamiento Computacional en Estudiantes de Ingeniería. Memorias de las 3º Jornadas sobre las prácticas docentes en la Universidad Pública. "El proyecto político académico de la Educación Superior en el contexto nacional y regional" (2020) 598-608.
   Recuperado de: http://sedici.unlp.edu.ar/handle/10915/111431
3. Jiménez Rey, E.: Computación en Ingeniería: La experiencia de pensar para solucionar problemas con algoritmos y programas en aula real y aula virtual, Anales de SAEI 2019 (48 JAIIO), Simposio Argentino de Educación en Informática (2019) 1-16.
   Recuperado de: http://sedici.unlp.edu.ar/handle/10915/88784
4. Jiménez Rey, E., Servetto A. y Calvo, P.: Experiencias de Educación Virtual en Tecnología: Algoritmia y Programación para Carreras de Ingeniería. En Escenarios y recursos para la enseñanza con tecnología: desafíos y retos, 01 | Innovación Docente. Barcelona: OCTAEDRO Editorial (2022) 161-169.
   Recuperado de: https://octaedro.com/libro/escenarios-y-recursos-para-la-ensenanza-con-tecnologia-desafios-y-retos/
5. Nickerson, R., Perkins, D. y Smith, E.: Enseñar a pensar. Barcelona: Paidós /M.E.C. (1987).
6. Jiménez Rey, E.: El Impacto del Pensamiento Computacional en el Diseño y la Codificación de Algoritmos. Actas del 8vo Congreso Nacional Ingeniería Informática / Sistemas de Información Virtual 2020 CONAIISI (2020) 630-636.
   Recuperado de: http://conaiisi2020.frsfco.utn.edu.ar/
7. Zapata Ros, M.: Pensamiento Computacional: Una nueva alfabetización digital. Revista de Educación a Distancia (2015) 46(4).
8. Wing, J.: Computational Thinking: What and Why? Magazine of Carnegie Mellon University's School of Computer Science (2010).

9. Simari, G.: Los fundamentos computacionales como parte de las ciencias básicas en las terminales de la disciplina Informática. VIII Congreso de Tecnología en Educación y Educación en Tecnología (2013).
Recuperado de: http://sedici.unlp.edu.ar/handle/10915/27579
10. Gómez, N.S.: Pensamiento computacional, innovación y perspectivas interdisciplinarias en ámbitos educativos (Tesis de maestría) (2020).
Recuperado de: http://sedici.unlp.edu.ar/handle/10915/111306