Fecha de recepción: diciembre 2023
Fecha de aceptación: enero 2023
Versión final: febrero 2024

Computación en Ingeniería: el diseño algorítmico para la codificación consciente

Elizabeth Jiménez Rey[1] y Arturo Servetto[2]

Resumen: La solución de problemas con programas conjuga a la Algoritmia y a la Programación, sin reducir la programación a una secuencia lógica de instrucciones. Ingeniar un algoritmo es idear una estrategia: identificar objetivo y recursos (datos y funciones) del programa, analizar casos posibles y sintetizarlos en un procedimiento. Desarrollar un programa es codificar la estrategia: refinar el problema en subproblemas (Prólogo o preparación de datos, Resolución o transformación de datos en resultados y Epílogo o informe de resultados). Se proporciona a los estudiantes de ingeniería un Modelo de Programa Tipo para estructurar y visibilizar en un texto el pensamiento computacional.

Palabras clave: Algoritmo - aprendizaje significativo - programación - solución - técnica de enseñanza.

[Resúmenes en inglés y portugués en la página 81]

[1] **Elizabeth Jiménez Rey.** Ingeniero Civil UNA/UBA. Especialista en Ingeniería de Sistemas en la FIUBA. Diplomada en Inteligencia & Pedagogía Compleja en la Multiversidad Mundo Real "Edgar Morin". Docente investigador en el Departamento de Computación de la Facultad de Ingeniería de la UBA (hasta el 31 de agosto de 2021).

[2] **Arturo Carlos Servetto.** Licenciado en Informática en la UNLP. Docente investigador y director del Laboratorio de Ingeniería de Datos y Educación en Tecnología del Departamento de Computación de la Facultad de Ingeniería de la UBA.

Desafío para enseñar y dificultad para aprender

Los autores, profesores responsables de la enseñanza, el aprendizaje y la evaluación de la asignatura Computación en cursos de la Facultad de Ingeniería de la Universidad de Buenos Aires, describen una experiencia educativa desarrollada durante el primer cuatrimestre del año 2021 en modalidad virtual y estructurada en dieciséis clases, una por semana, de cuatro horas de duración.

La asignatura es de formación básica por Resolución Ministerial del año 2001 y la pueden cursar los alumnos de carreras de Ingenierías al ingresar a la facultad luego de aprobar el Ciclo Básico Común, en simultaneidad con otras asignaturas de Ciencias Básicas muy demandantes de tiempo y esfuerzo como Álgebra, Análisis Matemático y Física. Para su aprendizaje se requiere desplegar el pensamiento computacional (creativo, lógico y crítico) para resolver problemas que en general resulta extraño o está poco o mal desarrollado en los estudiantes, que además consideran a la asignatura como ajena a sus intereses profesionales. Representa un desafío para los docentes despertar y mantener la motivación de los alumnos y evitar que deserten. La metodología educativa cobra un rol fundamental en este contexto con diversos condicionamientos desfavorables (Jiménez Rey, Servetto y Calvo, 2020).

Se aborda la enseñanza de Computación desde un enfoque procedimental que implica el *saber hacer* operativo (conocimiento técnico) y el *saber por qué se hace* formativo (conocimiento tecnológico). Los estudiantes tienen que idear una estrategia, es decir, ingeniar un algoritmo (una secuencia finita de pasos ejecutables para resolver el problema propuesto en un tiempo determinado) (Lerch y de Vedia, 2014). Constituye la primera dificultad a enfrentar. Y luego deben codificar la estrategia, es decir, representar el algoritmo en un lenguaje de programación. Constituye la segunda dificultad a enfrentar. Los pasos que conforman el algoritmo deben seguir un orden lógico y ser precisos, expresados en un lenguaje de programación de alto nivel como Python, para que la computadora pueda interpretar y ejecutar las instrucciones. La Algoritmia y la Programación no se aíslan, sino que se distinguen y se integran (Jiménez Rey, 2020).

Modelo de programa tipo y pensamiento computacional

Se utiliza el Modelo de Solución de Problemas del matemático húngaro George Pólya, aplicado al ámbito de la construcción de programas para desarrollar el pensamiento computacional en los estudiantes. El modelo consta de cuatro fases (Nickerson, Perkins y Smith, 1987): Análisis (comprender el problema), Diseño (diseñar una estrategia), Codificación (implementar la estrategia) y Evaluación (ejecutar el programa). El Análisis del problema y el Diseño de la solución, son las *fases algorítmicas* y constituyen el desafío creativo de aprendizaje. La Codificación del algoritmo y la Evaluación del programa son las *fases de programación*. En Computación, el *algoritmo* es el procedimiento que permite resolver un problema y el *programa* es la expresión del algoritmo en un lenguaje de programación para que la computadora pueda ejecutarlo.

Se proporciona a los estudiantes un dispositivo didáctico como modelo pedagógico denominado Modelo de Programa Tipo (MPT). Se trata de una plantilla, esquema o estructura, que constituye un andamiaje para que los estudiantes puedan organizar la solución del problema en un procedimiento algorítmico a partir del enunciado, hacer visible el pensamiento computacional y expresar el algoritmo en lenguaje Python. Este MPT es un archivo creado con el editor del entorno integrado de desarrollo y aprendizaje de Python denominado programa.py.

Zapata Ros (2015) sostiene que las competencias que se muestran como más eficaces en la codificación de un algoritmo son la parte más visible de una forma de pensar que es útil no sólo en el ámbito de las actividades cognitivas que intervienen en el desarrollo y en la creación de programas y de sistemas informáticos. 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 y que favorece las competencias computacionales. Según Wing (2006), 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 (2013) explicita que Wing describe esta forma de pensamiento como una combinación de las formas de pensamiento matemático e ingenieril (Jiménez Rey, 2019).
Para ayudar a los alumnos a resolver problemas de dificultad progresiva con programas, tanto en el proceso de enseñanza como en el proceso de aprendizaje, se favorece el desarrollo de este pensamiento específico mediante el análisis del problema y el diseño de algoritmos (fases Análisis y Diseño) como una competencia clave de aprendizaje en la formación de los estudiantes de ingeniería, sin reducir la construcción de programas al hallazgo de una secuencia de instrucciones (Fases Codificación y Evaluación). La utilización de la plantilla programa.py permite conjugar en un texto las fases algorítmicas y de programación que componen un programa.

Diseño algorítmico y codificación consciente

Los estudiantes, a partir del enunciado del problema:

- en la fase algorítmica, tienen que documentar el objetivo del programa, el análisis de casos posibles, la síntesis de casos para la generalización y los recursos del programa, a través de comentarios multilínea no ejecutables encerrados entre comillas (‘” … ‘”); y
- en la fase de programación, tienen que descomponer el problema en subproblemas, es decir, en un Prólogo o preparación de los datos, una Resolución o transformación de los datos en resultados y en un Epílogo o informe de resultados, a través de comentarios de una sola línea encabezados por el símbolo numeral (#). Para guiar el proceso de codificación del algoritmo, este primer nivel de descomposición se documenta en líneas de encabezado de código.

Se presenta un ejemplo de uso de la plantilla programa.py:

‘” Objetivo del programa (descripción del problema que resuelve): Solicita dos fechas parciales con formato aaaamm, la primera menor que la segunda, e informa la diferencia en años y meses entre ambas
Autor/es: 1C2021

Versión: 1
Fecha: 21/4/2021
Análisis de Casos (número de caso, estado inicial, transformación, estado final -se consideran casos base para la prueba del programa):
1. 202001, 202103 - 2021-2020=1 año, 03-01=2 meses - 1 año y 2 meses
2. 202004, 202103 - 2021-2020=1 año, 03-04=-01; año-1 y meses+12 - 11 meses
Síntesis de Casos (composición de casos para la generalización):
años=año2-año1
meses=mes2-mes1
si meses<0, entonces años=años-1 y meses=meses+12
Recursos (variables y funciones del programa -nombre, propósito, y si se transforman o no en el programa)
Datos que solicitar al usuario (sea en el prólogo o sea durante la resolución):
fecha_parcial_menor, fecha_parcial_mayor, enteros con forma aaaamm
Auxiliares (necesarios para transformaciones intermedias):
año1, año2, mes1, mes2, partes de las fechas parciales
Resultados (a informar sea durante el desarrollo o en el epílogo):
años, meses
Funciones propias (necesarias para la descomposición del problema en subproblemas):
no
‘”
# 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(‘Diferencia entre dos fechas parciales expresadas con año y mes’)
print()
# 1.1.2 Descripción o aclaraciones al usuario (opcional)
# 1.2 Datos iniciales
# 1.2.1 Solicitud e ingreso de datos desde teclado (opcional, si los datos se piden
# durante la resolución)
fecha_parcial_menor = int(input(‘Ingrese fecha menor expresada aaaamm:‘))
print()
fecha_parcial_mayor = int(input(‘Ingrese fecha mayor expresada aaaamm:‘))
print()
# 1.2.2 Establecimiento de valores iniciales para datos auxiliares o que se
# transformarán en resultados (opcional)
# Descomposición de fechas
año1 = fecha_parcial_menor // 100
mes1 = fecha_parcial_menor % 100
año2 = fecha_parcial_mayor // 100
mes2 = fecha_parcial_mayor % 100
#2 RESOLUCIÓN

```
# Descomposición del problema en subproblemas 2.x que a su vez pueden requerir
# ingreso o inicialización de datos o mostrar resultados
# Diferencia de años
años = año2 - año1
# Diferencia de meses
meses = mes2 - mes1
# ¿La diferencia de meses es negativa?
if meses < 0:
años = años - 1 # se puede simpificar la signación poniendo años -= 1
meses = meses + 12 # o también meses += 12
# 3 EPÍLOGO
# 3.1 Muestra de la solución del problema por pantalla (opcional, si sólo se muestran
# resultados durante la resolución)
if años == 0 or meses == 0:
if años != 0:
print (años, ‹año/s›)
else:
print(meses, ‹mes/es›)
else: # años != 0 and meses != 0
print(años, ‹año/s y›, meses, ‹mes/es›)
print()
# 3.2 Finalización 3.2.1 Impresión de la despedida al usuario en pantalla
print('Chau, chau, adios.')
# 3.2.1 (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 tecla Enter para terminar el programa...') # pausa forzada
```

Reflexiones finales

La tendencia de los estudiantes es desarrollar programas por prueba y error, secuenciando instrucciones del lenguaje Python, intentando que funcione (sea eficaz), sin tomar plena conciencia del problema que están resolviendo y en desmedro de la metodología que representa la plantilla, y por consiguiente de la claridad y eficiencia de los programas.
Se presentan testimonios de algunos estudiantes acerca de la contribución de la plantilla al aprendizaje de la solución de problemas con programas:

- *"La plantilla de programación fue muy útil y sirvió de guía para ordenarnos a la hora de formar el programa.*
- *"Lo usaba mucho cuando no entendía algo del código y necesitaba ver para qué servía (objetivo y análisis de casos), y cómo estaba adaptada la idea en lenguaje código (síntesis de casos)".*

- "Ayudaba en una lectura y visibilidad del código".
- "Ayuda con respecto a la organización y siento que también ayudó a aprender a pensar en una forma que nos beneficie a la hora de pensar en un algoritmo".
- "La plantilla fue fundamental para una buena organización en gestionar el programa y presentar un buen diseño".
- "La sección de síntesis de casos fue clave para poder pensar el algoritmo que debíamos realizar y luego pensar cómo codificarlo a partir de los conocimientos adquiridos. Si uno realiza a consciencia ese apartado y conoce los comandos necesarios puede tranquilamente codificar los algoritmos al programa Python".

Referencias bibliográficas

Jiménez Rey, E. (2019) *Computación en Ingeniería: La experiencia de pensar para solucionar problemas con algoritmos y programas en aula real y aula virtual.* Anales de SAEI 2019. Simposio Argentino de Educación en Informática (JAIIO), ISSN 2683-8958, pp. 1-16.

Jiménez Rey, E. (2020). *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, pp. 630-636. Recuperado de: http://conaiisi2020.frsfco.utn.edu.ar/*

Jiménez Rey, E., Servetto, A., y Calvo, P. (2020). *Desarrollo del 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", pp. 598-608. Recuperado de: http://sedici.unlp.edu.ar/handle/10915/111431

Lerch, C. y de Vedia, L. (2014) "El conocimiento tecnológico y el conocimiento ingenieril en la formación del ingeniero para un mundo cambiante". En: *La educación del ingeniero para un mundo cambiante*, pp. 49-78, 6, ANCEFN. Ciudad Autónoma de Buenos Aires: Luis de Vedia Editor.

Nickerson, R. S., Perkins, D. N. y Smith, E. E. (1987) *Enseñar a pensar*. *Aspectos de la aptitud intelectual.* Barcelona: Paidós.

Simari, G. (2013) *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. Santiago del Estero. Recuperado de: http://sedici.unlp.edu.ar/bitstream/handle/10915/27579/Documento_completo.pdf?sequence=1&isAllowed=y

Wing, J. (2006) "Computational Thinking". En: *Communications of the ACM*. Marzo. Vol. 49, n°. 3, 33-35.

Zapata Ros, M. (2015) *Pensamiento Computacional: Una nueva alfabetización digital*. RED Revista de Educación a Distancia, 46(4). Recuperado de: https://www.um.es/ead/red/46/

Abstract: Solving problems with programs combines Algorithms and Programming, without reducing programming to a logical sequence of instructions. Engineering an algorithm is devising a strategy: identifying the objective and resources (data and functions) of the program, analyzing possible cases and synthesizing them into a procedure. Developing a program is codifying the strategy: refining the problem into subproblems (Prologue or preparation of data, Resolution or transformation of data into results and Epilogue or report of results). Engineering students are provided with a Type Program Model to structure and make computational thinking visible in a text.

Keywords: Algorithm - meaningful learning - programming - solution - teaching technique.

Resumo: A resolução de problemas com programas combina Algoritmos e Programação, sem reduzir a programação a uma sequência lógica de instruções. Projetar um algoritmo é traçar uma estratégia: identificar o objetivo e os recursos (dados e funções) do programa, analisar possíveis casos e sintetizá-los em um procedimento. Desenvolver um programa é codificar a estratégia: refinar o problema em subproblemas (Prólogo ou preparação de dados, Resolução ou transformação de dados em resultados e Epílogo ou relatório de resultados). Os alunos de engenharia recebem um Modelo de Programa Tipo para estruturar e tornar o pensamento computacional visível em um texto.

Palavras chave: Algoritmo - aprendizagem significativa - programação - solução - técnica de ensino.

[Las traducciones de los abstracts fueron supervisadas por el autor de cada artículo]