Actas de las Jenui, vol. 4. 2019.
Páginas: 71-78

Aprendiendo a programar. Nuevos retos, nuevas propuestas

Brian Jiménez-García[1,2], Cristina Pérez Solà[1], Pau Andrio Balado[1], M. Jesús Marco Galindo[1]

[1]Estudis d'Informàtica, Multimèdia i Telecomunicació, Universitat Oberta de Catalunya
[2]Bijvoet Center for Biomolecular Research, Faculty of Science-Chemistry, Utrecht University
bjimenezga@uoc.edu, cperezsola@uoc.edu, pandrio@uoc.edu, mmarcog@uoc.edu

Resumen

La enseñanza de la programación en el ámbito universitario, a pesar de toda la experiencia acumulada, presenta muchos retos aún por alcanzar. Son diversos los elementos que añaden complejidad a este empeño y diversos también los estudios y propuestas didácticas que se proponen para abordarlos. El reto es aún mayor desde que, recientemente, se han incorporado al panorama universitario nuevos programas de especialidad con asignaturas de programación en ámbitos multidisciplinares como la bioinformática o la ciencia de datos.

En este artículo se describe la experiencia de dos asignaturas de nivel de máster: "Programación para la bioinformática" y "Programación para la Ciencia de Datos", ambas introductorias a la programación en el lenguaje Python, pero orientadas cada una de ellas a la resolución de los problemas específicos que se plantean en cada uno de estos dos ámbitos. Se trata de un objetivo tremendamente complejo, sobre todo si tenemos en cuenta el perfil heterogéneo de entrada de los estudiantes, poco o nada acostumbrados a la programación.

Abstract

In the university context and despite all the accumulated experience over the past decades, teaching computer programming is still challenging. The different approaches to accomplish this goal are diverse and complex, with many different didactic proposals. New challenges have aroused in recent times with the development of new and more specialized courses for multidisciplinary programs, such as bioinformatics or data science.

In this work we describe the experience obtained in two MSc programs: *Programming for Bioinformatics* and *Programming for Data Science*, both of them with an introductory aim at programming in the Python language and oriented to solve specific problems and challenges in the two different scopes. This is an extremely complex goal, considering the heterogeneous background of the students, not familiar with coding.

Palabras clave

Didáctica de la programación, Programación en otras disciplinas, Materiales interactivos, Notebooks, Python, aprendizaje situado

1. Motivación

Saber programar es una competencia imprescindible para cada vez más profesiones. Pongamos por ejemplo la ciencia de datos o la bioinformática que requieren un dominio importante de la programación para analizar cantidades ingentes de datos; en el primer caso para capturar, analizar y visualizar datos con fines concretos y, en el segundo, para trabajar con datos biológicos con la finalidad, por ejemplo, de diseñar nuevos fármacos.

La aparición de másteres especializados que responden a estas nuevas profesiones plantea cuestiones importantes sobre cómo debe ser la docencia de la programación: ¿servirán los cursos clásicos de algorítmica y programación de las tradicionales ingenierías informáticas o se necesita adquirir un nivel de competencia distinto? ¿El enfoque tradicional de enseñanza y el temario habitual serán adecuados para estos nuevos perfiles? ¿Y el lenguaje de programación, mejor uno de uso general o uno más alineado con el uso profesional? ¿El perfil de los estudiantes es homogéneo? Así que, si ya de por sí, enseñar (y aprender) a programar siempre ha sido una tarea complicada y con aún muchos retos por resolver como muy bien afirmaba Agustín Cernuda en "Todavía no sabemos enseñar a programar" [1], el panorama se complica aún más al plantear cómo hacerlo en estos nuevos másteres tan especializados.

Por tanto, diseñar una asignatura introductoria de programación para estos nuevos contextos supone un reto adicional que requiere de un cuidadoso diseño instruccional. Hay que determinar el enfoque de la asignatura, el lenguaje de programación, los recursos

didácticos, la infraestructura tecnológica necesaria para las prácticas y la metodología de enseñanza-aprendizaje. Sin perder de vista, además, los ya importantes retos que supone aprender a programar, entre otros, una curva de aprendizaje que suele ser inicialmente muy pronunciada y lenta, el alto nivel de abstracción que requiere por parte del estudiante y una tasa de abandono de las asignaturas elevada.

Por último, en nuestro caso, se añaden tres requisitos de contexto adicionales que son determinantes: la docencia se desarrolla en un entorno completamente virtual, bilingüe y donde deben coordinarse profesores de dos universidades distintas, la Universitat Oberta de Catalunya (UOC) y la Universitat de Barcelona (UB).

El presente artículo detalla una experiencia de dos asignaturas de máster: "Programación para la bioinformática" y "Programación para la ciencia de datos", ambas introductorias a la programación en lenguaje Python y, se estructura de la siguiente forma. La sección 2 introduce el contexto donde se enmarca la experiencia. La sección 3 describe la infraestructura tecnológica de las asignaturas, mientras que el desarrollo de cada una de ellas se detalla en la sección 4. Los resultados de la experiencia después de siete semestres de docencia en la primera asignatura con 396 estudiantes y tres semestres en la segunda asignatura con 328 estudiantes se analizan y se discuten en la sección 5, y finalmente, en la sección 6 se presentan las conclusiones del estudio.

2. Contexto de la experiencia

El curso 2015-2016 fue el primer curso en el que se impartió en los estudios de Máster Interuniversitario en Bioinformática y Bioestadística de la UOC/UB la asignatura de "Programación para la Bioinformática", asignatura optativa y de nueva creación que venía a cubrir el nicho de introducción a la programación para estudiantes con pocos o nulos conocimientos en la materia y con una formación y perfiles muy heterogéneos. Para contextualizar el problema, el perfil de los estudiantes del máster era (y sigue siendo) mayoritariamente de biología y ciencias de la salud: biología, bioquímica y biotecnología (52%), física, matemáticas y estadística (4%), química (4%), medicina, farmacia, enfermería (22%), ingeniería informática y telecomunicaciones (10%) y otros (8%).

Posteriormente, en el curso 2017-2018 se inició la asignatura de "Programación para la Ciencia de datos" como complemento de formación del Máster en Ciencia de datos, igualmente enfocada a estudiantes con pocos o nulos conocimientos de programación Python y con un perfil no ingenieril de base, con predominio de las ciencias sociales: únicamente el 34% proceden de titulaciones de ingeniería informática y afines, un 15% de telecomunicaciones, un 16% de matemáticas y física, un 18% de economía, dirección de empresas y similares y un 17% de otras titulaciones.

En el momento de emprender el diseño instruccional de las asignaturas, tuvimos presente en primer lugar, el ámbito profesional al que iban destinadas y, por ello, el resultado final son dos asignaturas similares pero distintas, una enfocada a la ciencia de datos y la otra a la bioinformática. En segundo lugar, el perfil de los estudiantes, que en ambos casos es heterogéneo y, en la mayoría de casos, sin apenas conocimientos previos de programación. Por último, el entorno virtual en el que se desarrolla la docencia. Estos tres factores determinan en gran parte el resultado final.

2.1. Metodología docente

Así pues, planteamos el diseño de estas dos asignaturas respondiendo a las siguientes preguntas:

- ¿Para qué necesitará programar un bioinformático? ¿Qué problemas deberá resolver programando? ¿Y un científico de datos, cómo tendrá que utilizar la programación para tratar con cantidades ingentes de datos? Esto nos lleva al objetivo 1, relacionado con la *utilidad en el contexto profesional.*
- **¿**Cuál es la mejor manera de aprender teniendo en cuenta el perfil heterogéneo de estudiantes y sus conocimientos previos? Para responder a esta pregunta se plantea el objetivo 2, relacionado con la *personalización del aprendizaje.*

Con estos objetivos en mente y sin perder de vista los condicionantes de nuestro entorno de enseñanza virtual, se diseñan las asignaturas.

Respondiendo al primer objetivo, se opta por una metodología docente inspirada en algunos principios de lo que se conoce como Aprendizaje situado [7] según el cual el aprendizaje se fundamenta en la resolución de los problemas a través de la aplicación en situaciones cotidianas del propio contexto a través de prácticas auténticas (en el sentido de relevantes para ese contexto), aumentando así el aprendizaje significativo.

Así pues, se opta por diseñar el aprendizaje a través del planteamiento de una serie de problemas prototípicos del ámbito que se van resolviendo proporcionando al estudiante los recursos y conocimientos y el acompañamiento necesario en cada momento. En el caso de la programación bioinformática, problemas prototípicos de análisis de

secuencias, utilizando cadenas de ADN, por ejemplo. En el caso de la programación para la ciencia de datos, problemas relacionados con alguna de las fases del ciclo de vida de los datos: desde su captura hasta la visualización, una vez ya procesados. Diseñar el programa a partir de los problemas prototípicos que se trabajan en el ámbito concreto también incide en un aumento de la motivación del estudiante, que ya desde el inicio encuentra sentido a lo que va aprendiendo y practicando, sabe para qué sirve y cuál es su aplicación profesional.

El enfoque del *aprendizaje contextualizado* promueve, además, la *práctica continuada* a base de proponer sucesivos ejercicios prácticos de programación a través de los que se introduce el contenido teórico necesario. Así pues, se parte de un problema contextualizado que se resuelve de manera práctica y que lleva al estudio y aprendizaje de los contenidos teóricos necesarios para su solución.

En relación al segundo objetivo, dado el perfil heterogéneo de entrada de los estudiantes a estos másteres, se plantean las asignaturas considerando que el estudiante no posee conocimientos previos de programación y que, muy posiblemente ha trabajado poco o nada con contenidos de tipos abstracto o matemático. Para ello, se define un conjunto de actividades de *evaluación continua* a desarrollar a lo largo del semestre que aseguran un aprendizaje progresivo de la programación y que el estudiante va desarrollando con el acompañamiento del profesor, que en este caso más que un trasmisor de conocimientos actúa de guía y acompañante durante el proceso, proporcionado un retorno individualizado de cada actividad. Así, el estudiante conoce cómo va progresando en cada actividad, qué debe corregir y qué debe mejorar. Es, en definitiva, una evaluación continua formativa.

Para mejorar aún más la atención personalizada y dar respuesta a los múltiples perfiles iniciales de los estudiantes, se optó por un *espacio de docencia integrado,* fusionando las aulas de teoría y laboratorio en una única aula. De este modo el estudiante tiene un único profesor de referencia que conoce su punto de partida y su evaluación y le ayuda en lo que necesita, ya sea porque no entiende un concepto algorítmico, porque no sabe encontrar la función necesaria en una librería determinada o porque no consigue instalar un software específico que necesita. Por otro lado, el hecho de utilizar una metodología de aprendizaje situada y práctica dificulta separar las actividades en teóricas y prácticas de laboratorio como se ha venido haciendo tradicionalmente en las asignaturas clásicas de programación.

En definitiva, la metodología docente se caracteriza por:

- Aprendizaje *contextualizado* en los problemas prototípicos del ámbito de conocimiento y eminentemente *práctico* donde los conocimientos teóricos necesarios se introducen a través de la práctica.
- *Evaluación continuada* complementada con un *retorno* (*feedback*) *individualizado* por parte del profesor para acompañar al estudiante en su proceso de aprendizaje.
- *Aula docente integrada* donde se desarrolla la práctica y se adquieren de manera simultánea los conceptos teóricos necesarios para llevarla a cabo.

2.2. La elección del lenguaje de programación Python

Nuestro objetivo principal era aprender a programar gracias a un lenguaje de programación, no pese al lenguaje, y sin centrarse en la parte algorítmica. En este contexto, la elección de Python nos resulta natural dada su inherente legibilidad de base, su sintaxis sencilla y una curva de aprendizaje rápida demostrada en otros programas de estudios. A su vez, tiene otras ventajas muy importantes que nos ayudan a cumplir nuestros objetivos: 1) se trata de un lenguaje interpretado que suaviza el problema de otros lenguajes compilados, que requieren un conocimiento más profundo de la arquitectura de un ordenador, conocimientos que dado el perfil de estudiantes no podíamos asumir, 2) una base de usuarios grande y muy activa y 3) el número de librerías disponibles, de gran madurez y activamente mantenidas por la comunidad.

2.3. Los recursos didácticos

Los recursos didácticos de acuerdo con la metodología docente de aprendizaje contextualizado y práctico, así como el lenguaje de programación escogido, son también peculiares en cuanto al contenido y al formato en el que se presentan.

En relación al *contenido*, se elaboran una serie de unidades didácticas que cubren todo lo que el estudiante necesitará para resolver las actividades continuadas que se plantean durante el semestre: desde la instalación del software a los conceptos de Python fundamentales, pasando por el conocimiento de las librerías y funciones adecuadas para resolver problemas del ámbito. Así, las dos asignaturas analizadas en la experiencia comparten un contenido inicial común pero luego difieren en la parte específica y aplicada del contexto que les es propio.

Los contenidos de la parte común a ambas asignaturas son los siguientes:

Contenido común:

- Unidad 1. Instalación y configuración del entorno de programación Python.

- Unidad 2. Breve introducción a la programación en Python. Sintaxis básica.
- Unidad 3. Conceptos avanzados de Python. Control de flujo. Funciones y estructuras lógicas.
- Unidad 4. Librerías científicas en Python.

Y a partir de aquí la parte final es específica de cada asignatura para ajustarla a las necesidades del contexto profesional de cada una de ellas.

Contenido específico Ciencia de datos

- Unidad 5. Captura de datos en Python.
- Unidad 6. Pre-procesado de datos en Python.
- Unidad 7. Introducción al análisis de datos en Python.
- Unidad 8. Visualización de datos en Python.

Contenido específico Bioinformática

- Unidad 5. Librerías para la representación gráfica: ADN, ARN, secuencias y motivos.
- Unidad 6. Biopython.
- Unidad 7. Testing y calidad del software.

Estos contenidos se articulan a través de un formato *xWiki* y unos *notebooks* que permiten, por un lado, que sea un contenido "vivo" y fácilmente *modificable* que se puede ir ampliando y adaptando a las necesidades que se vayan detectando. Y por otro, que sea un contenido *interactivo***,** que integra texto y código ejecutable Python en el mismo recurso. Esto facilita enormemente el poder adquirir los contenidos teóricos necesarios mientras se está practicando. Es, por tanto, una herramienta clave y fundamental en la implantación de la metodología diseñada que posibilita el aprendizaje contextualizado y práctico que se deseaba. Dada su relevancia, la infraestructura tecnológica de los recursos se describe con más detalle en el siguiente apartado.

En definitiva, el estudiante aprende lo que necesita para resolver problemas concretos de su futuro ámbito profesional, en lugar de aprender a programar de manera genérica con un propósito general como ocurre en un curso tradicional de programación de una ingeniería informática, por ejemplo.

3. Infraestructura tecnológica

La primera versión de los recursos didácticos para la asignatura de "Programación para la Bioinformática" en 2015 se concibe en formato híbrido: por un lado el aula virtual y por otro los contenidos. El aula virtual está formada por el plan docente, el foro de estudiantes y el tablón del profesor (como es habitual en el modelo pedagógico de la UOC). El contenido del curso de diseña por primera vez en formato Wiki, en concreto, mediante la plataforma xWiki[1]. Esto nos permitía ir enlazando y activando contenido según avanzaba la asignatura a lo largo del semestre y con una misión muy concreta: que los estudiantes progresaran de forma más o menos homogénea y motivándolos a escribir sus preguntas y dudas bien en el foro de la asignatura o mediante correo electrónico dirigido al profesor. Efectivamente, creemos que fue un gran acierto pues los estudiantes se sentían menos abrumados por un contenido que iban descubriendo poco a poco, más predispuestos a hacer sus preguntas en público, ya que el resto de los compañeros avanzaban a ritmos similares, y, a la vez, recompensados por su esfuerzo de una forma más rápida y directa.

El contenido de la asignatura está formado por dos componentes principales: la teoría y ejercicios en forma de *Jupyter Notebooks* y la máquina virtual para descargarlos desde un repositorio remoto GitLab[2] y ejecutarlos. En su día, esta arquitectura nos pareció la más adecuada ya que evitaba a los estudiantes la ardua tarea de instalar un entorno de programación (IDE, intérprete, librerías) y, de este modo, los focalizaba en la misión principal que era introducirlos a la programación. Creímos conveniente utilizar VirtualBox[3] como entorno de virtualización dado que era un software gratuito, con amplio soporte para diferentes arquitecturas y compatible con diferentes sistemas operativos. No podíamos estar seguros en ningún momento de la máquina y el sistema operativo que podían tener cada uno de los estudiantes, por lo que VirtualBox nos pareció la mejor elección. Para la máquina virtual, el sistema operativo virtualizado que escogimos fue una versión ligera de Ubuntu, Lubuntu, una distribución muy conocida y estable de GNU/Linux, con gran aceptación en la comunidad. Lubuntu nos permitió optimizar el tamaño de la máquina virtual, así como los recursos que utilizaba en caso de configuraciones de hardware más modestas.

A su vez, todas las librerías Python, así como el intérprete, estaban o ya preinstaladas o bien disponibles a pocos comandos mediante el gestor de paquetes *apt*. Otro software que instalamos de serie en la máquina virtual fue un navegador web, en nuestro caso Firefox, y las extensiones de *kernel* de VirtualBox para poder maximizar el tamaño de la pantalla, así como también compartir ficheros entre la máquina host y la virtual, copiar y pegar, etc.

Finalmente, y para que los estudiantes pudieran siempre trabajar con la última versión del material, instalamos en la máquina virtual el gestor de

[1] xWiki: https://www.xwiki.org
[2] GitLab: https://about.gitlab.com/
[3] VirtualBox: https://www.virtualbox.org/

versiones *git* y compartimos las diferentes unidades de teoría y ejercicios a través de repositorios en un GitLab institucional. Mediante un sencillo script, el estudiante es capaz de sincronizar, el material en su máquina virtual cuando el contenido es liberado y trabajar sobre él. Cuando finalmente tiene lista su entrega, el estudiante comprimía los *Jupyter Notebook* necesarios y es capaz de subirlos al campus virtual de la asignatura desde la propia máquina virtual.

Es interesante como el uso de *Jupyter Notebook* en el ámbito del aprendizaje de la programación se está extendiendo de una forma pronunciada e irreversible. Es una tecnología muy atractiva y visual de cara al estudiante, que ejecuta los ejemplos de forma interactiva, y que otros profesores han considerado también e incluso reportado en estas mismas jornadas [10].

4. Casos de aplicación

4.1. Programación para la Bioinformática

En el campo de la bioinformática, la ingeniería del software juega un papel relevante en dos ámbitos relativamente disyuntos: el desarrollo de grandes proyectos software y el prototipado rápido de pipelines mediante scripting. En muchas ocasiones el segundo acaba mutando en el primero, pero los recursos personales (número de desarrolladores, *expertise*) y temporales (desarrollo de producto vs nuevo método o pipeline de soporte para la publicación de un artículo científico) suelen ser radicalmente diferentes. En este curso nos centramos en el segundo escenario, en el que habitualmente un científico, con poco o nulo conocimiento en programación, necesita ensamblar su propio *pipeline* de análisis de datos de origen biológico o bien desarrollar pequeños algoritmos bioinformáticos. Es por ello por lo que la parte específica de la asignatura incluye teoría y ejercicios sobre cálculo matricial (vía la librería NumPy [11]), representación gráfica (Matplotlib [5]), algoritmia y biología estructural (Scipy [6]) y genómica computacional (BioPython [2]).

A su vez, creímos muy importante incluir los conceptos esenciales sobre calidad del software y buenas prácticas, pilares sobre los que se sustentan principios científicos tan importantes como son la reproducibilidad y la repetitividad. Para este propósito, incluimos una última unidad de *testing* y calidad del software, apoyada en el uso de pruebas unitarias vía la librería estándar *unittest*.

4.2. Programación Python para la ciencia de datos

De manera análoga, la "Programación para la ciencia de datos" abarca proyectos de software de muy diferente envergadura, desde pequeñas pruebas de concepto exploratorias o estudios que extraen conocimiento de los datos con unas pocas líneas de código, a proyectos a gran escala, que requieren de equipos de trabajo multidisciplinares para lograr este mismo objetivo. De nuevo, la asignatura se centra en pequeños proyectos de ciencia de datos, a menudo llevados a cabo por un único programador y sin grandes recursos de cómputo. En particular, se dedica a proporcionar herramientas de programación en Python que cubran todo el ciclo de los datos, de manera que el estudiante acaba el curso pudiendo llevar a cabo de manera autónoma pequeños proyectos de ciencia de datos de su interés.

Así pues, la parte específica cubre estrategias de captura de datos (entre las que se incluyen tanto la interacción con fuentes de datos de terceros a través de *Application Programming Interfaces* (API) como librerías específicas de *web crawling* como Scrapy[4], que permiten recolectar datos de cualquier web); librerías que ofrecen estructuras de datos y herramientas de tratamiento de datos sobre estas estructuras (pandas [8]); librerías que implementan algoritmos de aprendizaje automático (scikit-learn [9]); así como librerías de visualización de datos genéricas (matplotlib [5]) y especializadas para diferentes tipos de datos (por ejemplo, grafos con networkx [4] o mapas con geoplotlib [3]).

5. Resultados

En estos tres semestres experiencia en la asignatura de "Programación para la ciencia de datos", se constata un *perfil* de los estudiantes muy heterogéneo que se puede agrupar en tres categorías (Figura 1). Un primer grupo minoritario de estudiantes que programan habitualmente o que incluso conocen el lenguaje Python, un segundo grupo de alrededor de la mitad de estudiantes que saben algo de programación, aunque no programan habitualmente o ya hace mucho que hicieron un curso inicial y a un tercer grupo, también importante, de estudiantes que no tienen ningún conocimiento previo de programación. Esta disparidad de perfiles dificulta encontrar un enfoque que dé respuesta a todos ellos.

[4] Scrapinghub, Ltd. "Scrapy: A Fast and Powerful Scraping and Web Crawling Framework": https://scrapy.org/

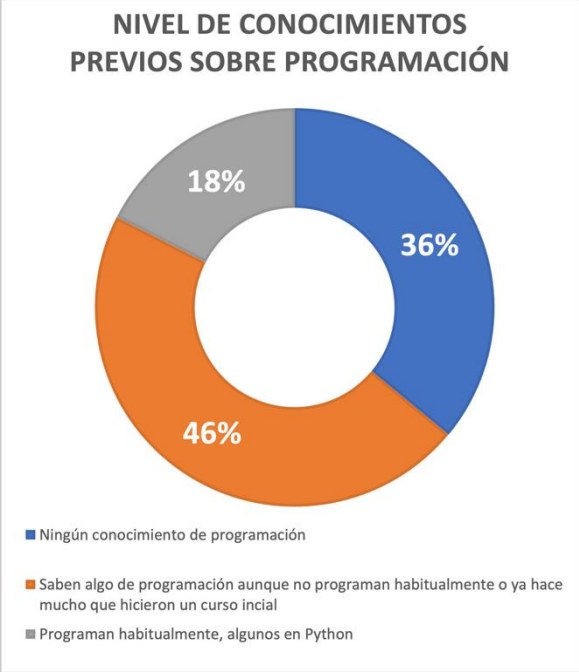

Figura 1 Nivel de conocimientos previos sobre programación de los alumnos matriculados para el 2º semestre del curso 2017-2018 y el 1º semestre del 2018-2019

Tras siete semestres de experiencia en "Programación bioinformática" y tres en "Programación para la ciencia de datos", se constata que los resultados académicos de esta propuesta son notables teniendo en cuenta la disparidad de perfiles de los estudiantes.

Los resultados de ambas asignaturas a lo largo de este periodo (Figura 2) confirman que el *rendimiento* (aprobados sobre matriculados en todos los semestres ha sido superior al 84% y el seguimiento de la evaluación continua siempre por encima del 87%, cifras muy superiores a las que se obtienen en un primer curso de programación genérico. Sin ir más lejos, en las asignaturas de "Fundamentos de programación" de los Grados de Ingeniería Informática y de Tecnologías de telecomunicación de la UOC, el rendimiento está habitualmente alrededor del 35%.

Programación para la bioinformática

Semestre	Numero Matriculados	Rendimiento	Seguimiento evaluación continua
2015-2016 1er. semestre	35	91.43%	97.24%
2015-2016 2o. semestre	36	88.89%	94.44%
2016-2017 1er. semestre	89	87.64%	87.64%
2016-2017 2o. semestre	73	93.15%	94.52%
2017-2018 1er. semestre	97	87.63%	92.75%
2017-2018 2o. semestre	66	93.94%	93.94%

Programación para la ciencia de datos

Semestre	Numero Matriculados	Rendimiento	Seguimiento evaluación continua
2017-2018 1er. semestre	62	82.3%	83.6%
2017-2018 2o. semestre	53	92.86%	92.86%

Figura 2 Porcentaje de rendimiento y seguimiento de la EC por semestre en cada una de las asignaturas.

Tras las tres primeras unidades didácticas (ejercicio y teoría) y aproximadamente un mes después del inicio del curso, la práctica totalidad de los alumnos sin ningún conocimiento previo de programación han alcanzado un nivel que les permite enfrentarse a retos más exigentes dentro del ámbito de las dos asignaturas. Si bien es cierto que para un estudiante avanzado y de perfil tecnológico que programa habitualmente, las primeras unidades no suponen un gran reto, se ha observado que éstas sirven para asentar conocimientos (a veces sorprendentemente nuevos para alumnos avanzados) y establecer una base común a partir de la cual, todo el grupo está preparado para enfrentarse a problemas reales relativos a la bioinformática o la ciencia de datos, terminando así la asignatura con un buen conocimiento general y estructurado del ámbito que se ha tratado. Prueba de que se consiguen equilibrar los distintos niveles iniciales es que, de los pocos abandonos que se producen (7,3%), prácticamente en ningún caso se alega como causa un salto insuperable de complejidad entre una unidad y la siguiente. En la gran mayoría de los casos, cuando los estudiantes son preguntados por el motivo del abandono de la asignatura, alegan que se debe a una la falta de tiempo debido a una combinación de factores persones y/o profesionales.

En relación a la *satisfacción* de los estudiantes, a partir de las encuestas institucionales que se realizan cada semestre y de la encuesta específica que realizamos desde las asignaturas se constata que la satisfacción global de las asignaturas es muy alta, siempre por encima el 90%. En relación a los recursos en formato notebooks (contenido y actividades) la satisfacción es también alta o muy alta (Figura 3a, 3b y 3c).

Así mismo, destaca la valoración muy positiva de la atención recibida por parte de los profesores. Esto se debe a que la ayuda y el retorno individualizado que se presta a los estudiantes en muchos casos es determinante para que éste pueda adquirir las competencias de programación y seguir el semestre.

Estos resultados positivos confirman como adecuadas la contextualización de la programación en el ámbito concreto, la metodología basada en la práctica continua con un *feedback* personalizado y, el uso de los notebook que permiten un aprendizaje interactivo.

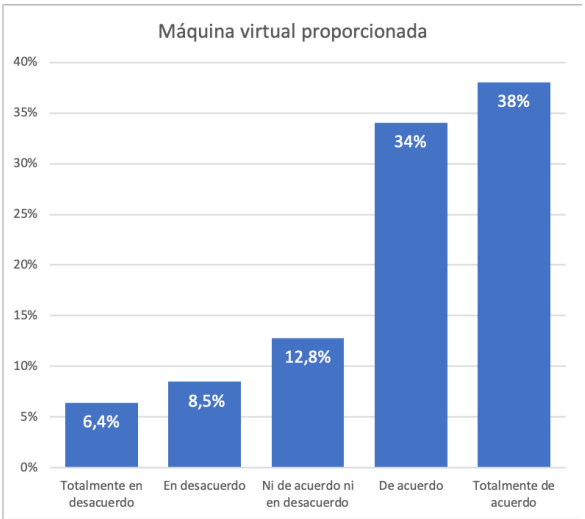

Figura 3a Resultados promedio de las encuestas de satisfacción en relación al hardware de "Programación para la ciencia de datos".

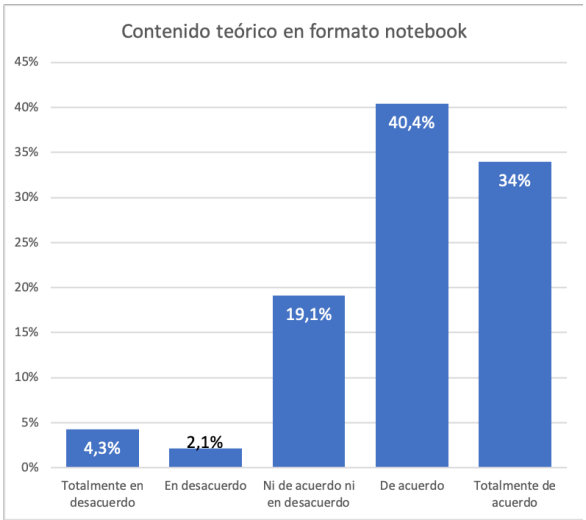

Figura 3b Resultados promedio de las encuestas de satisfacción en relación a las actividades de "Programación para la ciencia de datos".

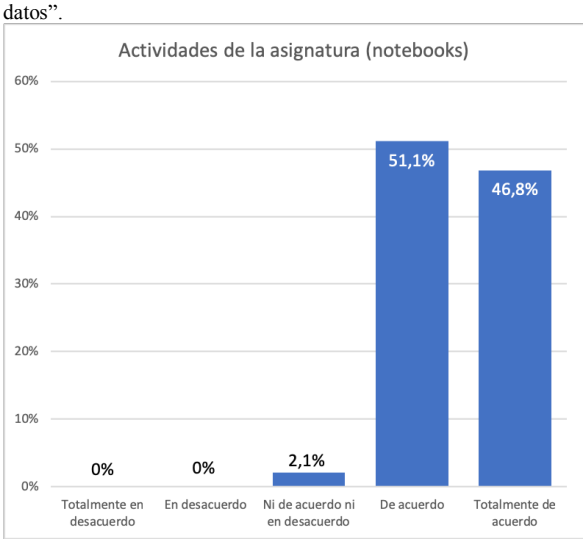

Figura 3c Resultados promedio de las encuestas de satisfacción en relación al contenido de "Programación para la ciencia de datos".

6. Discusión y trabajo futuro

Los resultados son muy positivos tanto respecto al rendimiento como a la satisfacción. Los comentarios que nos llegan de los estudiantes en las encuestas, algunos de los cuales se incluyen a modo ilustrativo, y que corroboran nuestra impresión después de estos semestres en las aulas, sugieren que:

- El modelo de evaluación basado en la realización de pequeños ejercicios de programación continuos es muy adecuado para mantener el ritmo de estudio y para conseguir un aprendizaje progresivo: *"La carga de trabajo es constante, se aprenden cosas muy útiles, las actividades tienen distintas dificultades para que una parte sea más mecánica y otra más de investigación*".
- El soporte continuo y el *feedback* por parte del profesor es clave e imprescindible: "… *el profesor ha sido el mejor de todas las asignaturas, respuestas rápidas, precisas, buen humor*", "*muchas gracias al equipo docente por su interés real y su actitud tan agradable siempre*"
- El uso de esta infraestructura tecnológica para los recursos es adecuado, aunque requiere de una inversión y explicación inicial un tiempo para que se aprenda a manejar bien: "*al principio me resultó un poco liado el uso de los notebooks para enviar actividades, quizás sería bueno detallar algo más los pasos a seguir*"
- El contenido es eminentemente práctico, pero hay que reforzar algunos aspectos de los recursos que quizás son demasiado escuetos par los estudiantes que no tienen conocimientos de programación previos en algunos aspectos teóricos. Recordemos que no se introduce la algorítmica y se aprende ya desde el inicio a programar en Python. Nos proponemos añadir más ejemplos comentados introductorios en cada unidad y vídeos tipos *screencast* para detallar más paso a paso cómo utilizar el software y resolver los primeros ejercicios prácticos. "*el contenido es factible, pero a la vez reta según qué ejercicios tengamos para la entrega*", *"Me parece que la asignatura está bien en cuanto al apartado práctico, pero daría más información sobre la parte teórica", "la teoría es bastante escueta pero suficiente para resolver las actividades*".
- El contenido contextualizado resulta muy bien valorado. Permite a los estudiantes poner en práctica en su entorno profesional de manera inmediata y ver la utilidad directa en el máster que van a cursar *"De todas las asignaturas*

cursadas es de las más próximas a lo que se espera en un máster de ciencia de datos" "excelente materia, nunca había trabajado tanto en Python. Aprendí muchas cosas valiosas para mi vida laboral y profesional"

En relación al trabajo futuro nos plateamos algunas mejoras que consideramos importantes:

- En el momento en el que se diseñó la plataforma tecnológica basada en máquinas virtuales, no existían (o no eran suficientemente maduras) los servidores Notebook o servicios online que proporcionaran acceso a recursos de tipo Jupyter Notebook. Esto parece haber cambiado con la aparición de servicios como Colaboratory de Google, o la madurez de Jupyter Notebook Server. Podría ser un objetivo realista a medio plazo sustituir el uso de máquinas virtuales por estos servicios en línea, eliminando así el componente de configuración de otro recurso adicional para poder ejecutar los notebooks de teoría y práctica. Google Colaboratory, de hecho, ya viene de serie con las librerías necesarios para ambas asignaturas. Por otro lado, esto nos limitaría la flexibilidad, ya que dependeríamos de un proveedor externo. Por lo tanto, nos enfrentamos a un dilema interesante en cuanto a si externalizar o no la plataforma en la que se ejecuta el material didáctico.
- El segundo reto que se plantea es crear un repositorio común en un espacio tipo Git para ofrecer estos recursos a cualquier estudiante de estos masters que lo necesite durante su titulación, ya sean tanto estudiantes que hayan cursado la asignatura en el pasado, como estudiantes que por su formación ya técnica no cursen estas asignaturas (pero que quizás no estén familiarizados con las librerías que se trabajan en ellas) "*Me ha gustado la asignatura y pienso que los puntos que se han visto pueden ser útiles en el desarrollo de máster*". El reto, en este caso, consiste en ofrecer todo el material de la asignatura en abierto (notebooks, máquina virtual, y material en formato xwiki), para que cualquier estudiante pueda seguirla de manera autónoma, a la vez que se ofrece la asignatura con evaluación continuada para los estudiantes matriculados. La externalización del material didáctico tendría también implicaciones en cuanto al despliegue de la versión abierta de las asignaturas.
- Diversos estudiantes nos piden más recursos para avanzar en Python "*Lo único que me da pena es que no haya un Python II porque me parece que ha sido la mejor asignatura*". Así que estamos ya diseñando una nueva asignatura programación avanzada en Python que pondremos en marcha el próximo curso. Esta nueva asignatura incluirá tanto ampliaciones de algunos temas que ya se tratan en las asignaturas actuales de manera más básica (por ejemplo, estructuras de datos avanzadas y uso avanzado de funciones), como nuevos temas que por su complejidad no se llegan a cubrir (interacción con el sistema operativo, optimización de algoritmos, análisis de rendimiento o *profiling*, programación paralela, y mantenimiento de aplicaciones).

Referencias

[1] Agustín Cernuda del Río. Todavía no sabemos enseñar programación. En *ReVisión 10,1. 2017.*

[2] Peter JA Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics." Bioinformatics 25, no. 11 (2009): 1422-1423.

[3] Andrea Cuttone, Sune Lehmann, y Jakob Eg Larsen. "geoplotlib: a Python Toolbox for Visualizing Geographical Data." arXiv preprint arXiv:1608.01933 (2016).

[4] Aric Hagberg et al. "Exploring network structure, dynamics and function using NetworkX." No.LA-UR-08-05495. Los Alamos National Lab. Los Alamos, NM. (2013).

[5] John D. Hunter, "Matplotlib: A 2D graphics environment." Computing in science & engineering 9, no. 3 (2007): 90-95.

[6] Eric Jones, Travis Oliphant, y Pearu Peterson. "{SciPy}: open source scientific tools for {Python}." (2014).

[7] J. Laver y E. Wenger. Situated learning: legitimate peripheral participation. Cambridge: Cambridge University Press. 1991.

[8] Wes McKinney. "Pandas: a foundational Python library for data analysis and statistics." Python for High Performance and Scientific Computing (2011): 1-9.

[9] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python." En *Journal of machine learning research* 12. Oct (2011): 2825-2830.

[10] José A. Troyano, Fermín Cruz, Mariano González, Carlos G. Vallejo, y Miguel Toro. "Introducción a la Programación con Python, Computación Interactiva y Aprendizaje Significativo." En *Actas de las Jenui 2018*: 223-230.

[11] Stéfan van der Walt, S. Chris Colbert, y Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation." Computing in Science & Engineering 13, no.2 (2011): 22-30.