Actas de las Jenui, vol. 3. 2018.
Páginas: 223-230

Introducción a la Programación con Python, Computación Interactiva y Aprendizaje Significativo

José A. Troyano, Fermín Cruz, Mariano González, Carlos G. Vallejo, Miguel Toro
Departamento de Lenguajes y Sistemas Informáticos
ETS Ingeniería Informática
Universidad de Sevilla
Av. Reina Mercedes s/n, 41012 Sevilla
troyano@us.es

Resumen

En este trabajo presentamos una experiencia docente, y el recurso docente correspondiente, que hemos seguido durante el primer cuatrimestre del curso 17/18 en una asignatura de introducción a la programación. Nuestro principal objetivo era trabajar sobre la motivación de los alumnos, haciendo que el aprendizaje de los fundamentos de la programación fuese lo más atractivo e intuitivo posible. Para ello, nos hemos apoyado en tres elementos principales: el lenguaje de programación Python, el uso de un entorno de desarrollo interactivo como los *notebooks* de Jupyter, y la adopción de una estrategia de aprendizaje significativo entendido como el proceso en el que en todo momento se sabe "para qué sirve" el conocimiento que se está adquiriendo. Los tres elementos han contribuido a mejorar la calidad de la enseñanza con respecto a ediciones anteriores de la asignatura. Los resultados académicos obtenidos en la evaluación y los indicios que hemos podido extraer de una encuesta realizada a los alumnos son muy positivos, lo que nos anima a seguir en esta dirección. Todo el material docente utilizado en la asignatura está disponible públicamente en un repositorio GitHub.

Abstract

In this paper we present a teaching experience, and the corresponding teaching resource, which we have followed during the first semester of the 17-18 academic year in a subject of introduction to programming. Our main objective was to work on the motivation of the students, making the learning of the main concepts of programming as attractive and intuitive as possible. We have relied on three main elements: the Python programming language, the use of an interactive development environment such as the Jupyter *notebooks*, and the adoption of a meaningful learning strategy. We understand by meaningful learning the process in which at every moment the student knows "what is the use" of the knowledge that is being acquired. The three elements have contributed to improve the quality of the teaching compared to previous editions of the subject. The academic results obtained in the evaluation and several insights that we have been able to extract from a survey made to the students are very positive, which encourages us to continue in this direction. All the teaching material used during the course is publicly available in a GitHub repository.

Palabras clave

Introducción a la programación, Python, Computación interactiva, Notebook, Aprendizaje basado en proyectos

1. Motivación

Aprender a programar es una tarea difícil, pero, ¿se puede hacer aún más difícil? Desgraciadamente, la respuesta es sí. Como dice una popular canción de los años ochenta: *la vida es complicada, ... y tú la quieres complicar más*. Algo de esto ocurre cuando se enfoca la enseñanza de esta actividad compleja y creativa desde una perspectiva que no ayuda en lo primero (la complejidad) y no potencia lo segundo (la creatividad).

En nuestra opinión, uno de los principales motivos por el que aparece esta innecesaria complejidad adicional es plantear una asignatura de introducción a la programación a partir de unos contenidos que reproducen el esquema típico de un manual de un lenguaje de programación. Porque, ¿alguien ha leído alguna vez, de principio a fin, uno de esos libros? Lo normal es que se usen como libros de consulta y que se salte de un punto a otro en busca de la información que en cada momento hace falta. Y si lo normal es usar los manua-

les de esta forma, ¿tiene sentido hacer que alumnos que no tienen conocimientos de programación tengan este tipo de esquema como guía de aprendizaje?

Hagamos, por un momento, el ejercicio de imaginar qué siente un alumno durante las primeras semanas de curso ante un hipotético plan de trabajo. Los contenidos para esas semanas serían:

1. ¿Qué es un lenguaje de programación?
2. Constantes, variables y tipos básicos
3. Operadores y expresiones
4. Instrucciones y estructuras de control
5. Funciones
6. Clases y objetos
7. ...

En clase irían recibiendo explicaciones de estos conceptos (algunos de ellos mucho más abstractos y difíciles de lo que un programador con experiencia pueda pensar) y podrán comprobar su funcionamiento con ejemplos simples. Como al principio no se conocen muchos elementos del lenguaje que permitan hacer cosas interesantes (no vamos a empezar con ficheros, ¿no?), los ejemplos *simples* más socorridos suelen ser funciones matemáticas como fibonacci, calcula_pi, es_primo, mcm, MCD...

Ante este panorama, un alumno sin conocimientos y sin una motivación especial puede empezar a consolidar las siguientes impresiones:

- Esto de programar es muy difícil, y cada vez se complica más.
- No tengo claro para qué sirve lo que me están explicando.

Si es así, mal inicio hemos tenido. De hecho, las impresiones que nos gustaría que los alumnos tuviesen en esas primeras semanas son:

- Esto de programar es muy interesante, y cada vez me lo parece más.
- Hay un montón de cosas que se pueden hacer con lo que estoy aprendiendo.

Esto es fácil pedirlo, pero ¿cómo conseguirlo? Nosotros no tenemos, en absoluto, la respuesta definitiva a esa pregunta. Pero, a través de nuestra propia experiencia y errores de planteamiento pasados, hemos identificado cosas que creemos que se pueden mejorar. Este trabajo es, precisamente, producto de ese intento de mejora, y en él explicamos los cambios que hemos afrontado en una asignatura de introducción a la programación durante el primer cuatrimestre del curso 17/18.

El resto del trabajo se organiza de la siguiente manera: en el apartado 2 explicamos las ideas principales de nuestra propuesta, el apartado 3 presenta el material docente que hemos usado, en el apartado 4 comentamos los aspectos metodológicos y analizamos los resultados de nuestra experiencia; por último, en el apartado 5 resumimos las conclusiones de nuestro trabajo.

2. Los elementos

En este apartado presentamos brevemente los tres ejes principales de nuestra propuesta de mejora de la asignatura.

2.1. Python

Python [9] es un lenguaje de programación interpretado en el que prima, fundamentalmente, la legibilidad del código. Tanto es así, que en la comunidad de desarrolladores se usa el término *pitónico* para denominar a aquellos programas Python que por su claridad y elegancia encarnan la filosofía del lenguaje. Esta filosofía está resumida en el famoso catálogo de recomendaciones PEP-20 también conocido como el "Zen de Python", una de cuyas principales recomendaciones es *readability counts*. Python es un lenguaje multiparadigma en el que conviven de forma nativa aspectos imperativos, funcionales y orientados a objetos. Estos paradigmas están muy bien desacoplados, lo que permite que la entrada al lenguaje se pueda hacer de forma progresiva empezando, por ejemplo, con un estilo imperativo e incluyendo posteriormente elementos funcionales y orientados a objetos.

Python cuenta con una amplia biblioteca estándar (filosofía *batteries included*) con herramientas para resolver muchas tareas de utilidad como, por ejemplo, la lectura de ficheros en distintos formatos (JSON, XML, CSV...), el desarrollo simple de interfaces gráficas, la conexión con bases de datos, y un largo etcétera. Además de la biblioteca estándar, hay una gran cantidad de bibliotecas desarrolladas por terceros que se organizan en el repositorio PyPI (*Python Package Index*). En la actualidad hay más de 120.000 paquetes en PyPI y subiendo a un ritmo de unos 1.000 paquetes al mes.

Si unimos el apoyo que suponen la biblioteca estándar y PyPI, con la claridad y legibilidad del lenguaje, tenemos un entorno en el que con muy pocas líneas de código se pueden hacer cosas interesantes. Por ejemplo, uno de los primeros proyectos que desarrollamos en clase consistió en:

- Leer varios ficheros con las cotizaciones bursátiles de varias empresas.
- Crear un diccionario de listas para representar la información leída.
- Seleccionar algunas empresas y generar una gráfica en la que se muestre una curva con la evolución de los precios de cada una en un período de tiempo.

La solución se organizó en 4 funciones y requirió escribir solo 18 líneas de código (muy legibles). Esta es la carta de presentación a la programación que Python nos permite escribir para nuestros alumnos, y es la razón por la que apostamos por él como lenguaje en una asignatura de introducción a la programación [5].

2.2. Aprendizaje significativo

El aprendizaje significativo [1] (del inglés *meaningful learning*) es un tipo de aprendizaje en el que el nuevo conocimiento adquirido por el alumno se conecta con otros conocimientos previos. Es un proceso constante de reajuste en el que los nuevos conocimientos se estructuran en función de los conocimientos previos y, a su vez, estos últimos se reestructuran con la nueva información recibida. Esta forma de aprender es aplicable a muchas áreas de conocimiento [7], [4], y es especialmente apropiada para contextos tecnológicos [2], [12], [11].

En el aprendizaje significativo es fundamental el "significado" del aprendizaje, es decir "para qué" sirve lo que se está aprendiendo. La búsqueda del significado es algo innato para nosotros y es un instrumento de aprendizaje muy potente si somos capaces de ponerlo a nuestro servicio. De alguna forma se trata de un mecanismo de uso eficiente de recursos: el ser humano tiene la tendencia a aprender "de verdad" solo aquello que considera útil. Esto, que usado de forma positiva puede potenciar el aprendizaje, en términos negativos (que el alumno no tenga claro para qué sirve lo que está aprendiendo) tiene unos efectos nefastos.

Además de la relación entre conocimiento previo y nuevo, hay un tercer elemento clave en el aprendizaje: la motivación del alumno. La motivación es el principal resorte para que el cerebro invierta energía en establecer conexiones entre lo que ya se conoce y lo que se va a aprender, y sin ella difícilmente se alcanzará un aprendizaje sólido.

En una asignatura de introducción a la programación, el nuevo conocimiento a adquirir lo constituye el lenguaje de programación que se va a descubrir y la capacidad para construir soluciones a partir de este lenguaje. Pero, ¿qué hay de los otros dos elementos del aprendizaje significativo: el conocimiento previo y la motivación? Nuestra propuesta pasa por cuidar bien el primero (conocimiento previo) para potenciar el segundo (motivación), y lo haremos mediante una estrategia de aprendizaje basada en proyectos. En este contexto uno de los conocimientos previos que más nos interesa es el del dominio de los proyectos, de manera que sea fácil (aún sin tener experiencia en programación ni conocer el lenguaje) comprender qué es lo que se va a construir. Si eso se transmite bien, tenemos ya el primer paso que necesita el aprendizaje significativo: saber para qué sirve lo que se va a aprender. En cuanto a la motivación, va a ser mucho mayor si, además, los proyectos están relacionados con temáticas cercanas al alumno. Por ejemplo, en nuestro caso el proyecto que mejor ha funcionado con los alumnos ha sido uno que se basa en el análisis de los *logs* de *Whatsapp* para extraer patrones de comportamiento de un grupo de *chat*.

2.3. Computación interactiva

El esquema más simple de computación interactiva viene dado por los entornos REPL (*Read Eval Print Loop*). Un entorno REPL proporciona una interfaz que lee instrucciones escritas por un usuario, las evalúa y muestra sus resultados. Por ejemplo, una consola de un sistema operativo es un entorno REPL que permite interactuar con el sistema. Con el tiempo este tipo de sistemas ha evolucionado, y en la actualidad existen muchas alternativas con interfaces mucho más intuitivas que hacen que esta forma de programar ofrezca muchas ventajas en ciertos contextos.

Uno de los proyectos de computación interactiva más consolidados es Jupyter [8] (continuación del proyecto IPython [6]) que se apoya en el concepto de *notebook* heredado de otros entornos de computación interactiva como Mathematica, Maple o Sage. Un notebook de Jupyter es un fichero JSON que permite integrar código y texto con formato (con imágenes, vídeos, HTML, LaTeX...). Un notebook tiene la doble ventaja de ser un documento con un formato cómodo para ser leído por una persona, y al mismo tiempo los fragmentos de código se pueden ejecutar. La unidad básica de los notebooks es la *celda*, que puede ser de dos tipos principales: *code* para los fragmentos de código y *markdown* para el texto con formato y otros contenidos.

El entorno de trabajo para los notebooks de Jupyter es una interfaz web que permite crear y editar de manera cómoda los notebooks, y que se conecta con el intérprete del lenguaje de programación a través de lo que se denomina *kernel*. El *kernel* por defecto es el de Python, pero cada vez hay disponibles más *kernels* para otros lenguajes de programación (R, Haskell, Julia, Ruby, Scala, Java...).

Para nosotros, los notebooks de Jupyter han sido todo un descubrimiento como medio para crear material docente, ya que permiten integrar lo mejor de los apuntes, transparencias y código ejecutable en un único recurso. Básicamente nos han servido para dos funciones principales:

- Para dar soporte a nuestro enfoque orientado a proyectos, mediante lo que hemos llamado *notebooks de ejercicios* en los que se intercalan explicaciones y celdas de código que hacen que el propio recurso sirva de guía para que los alumnos vayan desarrollando la solución al proyecto.

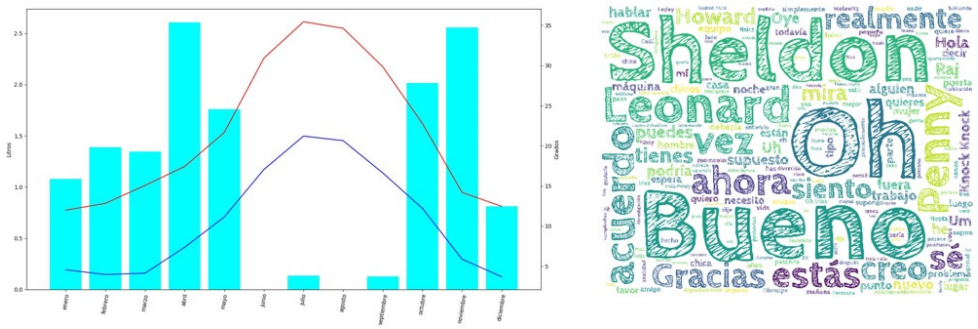

Figura 1: Gráficos generados en los proyectos *Clima y Whatsapp*

- Para crear tutoriales interactivos (los hemos denominado *noteboks de teoría*) que permiten dar a conocer el lenguaje a los alumnos con un recurso en el que directamente pueden ejecutar los ejemplos que ilustran los distintos elementos que están aprendiendo.

En el siguiente apartado explicaremos más en detalle las características del material docente que hemos preparado.

3. El material

Hemos creado tres tipos de contenidos que se diferencian en dos aspectos: el entorno de trabajo para los que están diseñados, y la función que pretendemos que cumplan dentro del proceso de aprendizaje de los alumnos. Los tres tipos de recursos son:

- Notebook de teoría: tutorial no exhaustivo que introduce distintos elementos del lenguaje Python.
- Notebook de ejercicios: proyecto desarrollado sobre un notebook Jupyter.
- Proyecto de laboratorio: proyecto desarrollado con el entorno Eclipse.

Todo el material está disponible públicamente en un repositorio GitHub.[1]

3.1. Notebooks de ejercicios

Es el recurso que mejor encarna nuestra propuesta, ya que integra los tres elementos que hemos presentado en el apartado 2: Python, computación interactiva y aprendizaje significativo. En el cuadro 1 se enumeran los proyectos disponibles en este momento en este formato.

Es el material usado en las clases de teoría. Gran parte de los alumnos traen a las clases de teoría ordenadores portátiles, lo que permite que dichas sesiones sean más dinámicas y que se pueda probar *in situ* las propuestas de solución que se van exponiendo en clase.

[1] https://github.com/fupus

Título	**Descripción**
Audiencias	Análisis de datos de audiencias televisivas
Bicicletas	Identificación de estaciones en una red de alquiler de bicicletas
Bolsa	Análisis de datos bursátiles
FutElo	Sistema de puntuación Elo sobre resultados de fútbol
ATP	Comparación y evolución de rankings con la métrica de Kendall
Recomendación	Sistema de recomendación de películas
Whatsapp	Cálculo de indicadores a partir de logs de conversaciones de Whatsapp
Montecarlo	Aplicaciones a la estimación de π y la fuerza de una jugada de póker

Cuadro 1: Notebooks de ejercicios

Los proyectos incluidos en los notebooks de ejercicios están parcialmente resueltos, e incluyen:

- La organización del proyecto en funciones. Para cada función se definen los parámetros y se incluye un comentario explicativo de su funcionamiento.
- Una celda de código con el test para cada función.
- La decisión de las estructuras usadas para representar los datos del proyecto.

No se incluye la implementación de las funciones, que es el grueso del trabajo a realizar en clase. El cuerpo de las funciones contiene solo una instrucción *pass* que deberá ser sustituida por una implementación que cubra lo que se indica en el comentario explicativo.

Es muy importante que estos proyectos sean motivadores para los alumnos ya que con ello nos aseguramos su interés y reforzamos el aspecto del *aprendizaje significativo* que hemos comentado en el apartado 2.2. En una encuesta que hemos realizado (cuyos resultados resumimos en el apartado 4.2) queda patente este hecho: los tres proyectos mejor valorados por los alum-

nos son, por este orden, *Whatsapp*, *Recomendación* y *Bicicletas*. Claramente son los que mejor conectan con sus intereses y con los dominios de aplicación que ellos más conocen.

Otro aspecto que consideramos muy útil desde el punto de vista de la motivación es que los resultados de los proyectos sean lo más atractivos posible. No es lo mismo producir como salida un listado de valores en una pantalla negra, que un gráfico que muestre de forma intuitiva relaciones entre datos o resultados de un determinado análisis. En la figura 1 se muestran dos ejemplos de visualizaciones generadas.

En el apartado de la generación de gráficos nos hemos apoyado fundamentalmente en *Matplotlib*, una de las bibliotecas Python más populares para visualización. *Matplotlib* es una biblioteca muy compleja, y hemos salvado esta complejidad proporcionando en cada momento a los alumnos las *instrucciones matplotlib* necesarias para generar las gráficas. Como trabajo del alumno queda la implementación de las instrucciones que construyen las estructuras de datos que requiera cada visualización. Este esquema de trabajo tiene la ventaja de que los alumnos van conociendo las posibilidades de una herramienta compleja, sin tener que enfrentarse aún a comprenderla en profundidad. Para los alumnos más avanzados y curiosos supone, además, una puerta abierta a explorar por su cuenta.

3.2. Proyectos de laboratorio

Los proyectos de laboratorio son muy parecidos a los notebooks de ejercicios, pero cumplen una función complementaria. Al igual que los notebooks de teoría, están parcialmente resueltos y organizados en funciones, pero están disponibles en *scripts* Pyhton (ficheros ".py") y están pensados para editarse con un IDE que, preferiblemente, integre el intérprete de Python[2]. En el cuadro 2 se enumeran los proyectos disponibles en este momento en este formato.

Es el material usado en la clase de laboratorio. El objetivo de estas sesiones es que los alumnos se familiaricen con un entorno de desarrollo de programas distinto al de los notebooks. Además, el rol del profesor cambia. Mientras que en las clases de teoría el profesor lleva el peso de la explicación y construcción de la solución, en las de laboratorio los alumnos asumen más responsabilidad. El profesor es un apoyo para los problemas que puedan surgir a los alumnos, pero son ellos los que tienen que crear la solución de los proyectos.

[2]Nosotros hemos usado Eclipse+PyDev, pero hay muchas más alternativas.

Título	**Descripción**
Mínimos Cuadrados	Cálculo de una recta de regresión mediante la técnica de mínimos cuadrados
Cifrado	Cifrado y descifrado de textos mediante el código de desplazamiento de César
D'Hont	Análisis de datos electorales y cálculo de escaños mediante la ley D'Hont
Vecinos	Clasificación automática basada en el método de los vecinos más cercanos
Imágenes	Funciones de manipulación de imágenes RGB (reflejo, rotación, transformaciones del color...)
Clima	Análisis de datos históricos de clima y generación de climogramas

Cuadro 2: Proyectos de laboratorio

3.3. Notebooks de teoría

Cada notebook es un tutorial (no exhaustivo) que introduce distintos elementos del lenguaje Python. La mayor parte de las celdas de código están completas, por lo que los notebooks son fáciles de seguir sin atascarse. El cuadro 3 muestra el listado de los notebooks disponibles.

En un primer momento pensábamos en una aproximación orientada a proyectos *pura* en la que, desde el minuto cero, se empezasen a programar soluciones completas. Pero pronto llegamos a la conclusión de que una aproximación tan *radical* no iba a funcionar por la cantidad de interrupciones que provocaría, en el primer proyecto, la continua aparición de nuevos elementos desconocidos para el alumno.

Nos encontrábamos en una típica situación de *la gallina y el huevo*: ¿qué hacer?, ¿explicar primero los elementos del lenguaje *a secas* y luego empezar con los proyectos? o ¿empezar por los proyectos y explicar *sobre la marcha* los elementos del lenguaje? Los notebooks de teoría nos dieron una salida: hacerlo en paralelo. Al ser recursos fáciles de seguir, pasaron a ser material de trabajo en tiempo de estudio que el alumno hace por su cuenta.

Tanto en clase de teoría como en laboratorio se puede reforzar constantemente este trabajo de descubrimiento del lenguaje con explicaciones específicas a demanda de lo que requiera cada proyecto o a raíz de preguntas de los alumnos. Pero esto es solo eso, un refuerzo: la toma de contacto con los elementos lenguaje cae de parte del alumno, con la ayuda de un material interactivo y fácil de utilizar. Un esquema de *flip teaching* que transfiere un determinado aprendizaje fuera del aula [3].

Número	Descripción
1	Introducción a Python
2	Expresiones y tipos predefinidos
3	Condicionales y bucles
4	Funciones
5	Secuencias, listas y tuplas
6	Diccionarios y conjuntos
7	Entrada y salida
8	Expresiones regulares

Cuadro 3: Notebooks de teoría

Figura 2: Perfil del alumnado

4. La experiencia

La experiencia docente que presentamos en este trabajo se ha puesto en marcha en una asignatura de introducción a la programación de dos grados impartidos en la ETS de Ingeniería Informática de la Universidad de Sevilla:

- GII-TI: Grado en Ingeniería Informática, Tecnologías Informáticas
- GISA: Grado en Ingeniería de la Salud

Antes de esta experiencia, en la asignatura se usaba un enfoque distinto al que planteamos en este trabajo, guiado por la presentación ordenada de los distintos elementos del lenguaje. En el grado GISA se usaba C y en GI-TII se usaba Java.

En este apartado aportaremos datos sobre el desarrollo de nuestra experiencia docente durante el curso 17/18. Empezaremos por comentar algunos detalles sobre la metodología seguida en las aulas y en la evaluación. En segundo lugar, mostraremos algunos indicios sobre la opinión de los alumnos a partir de los datos obtenidos con una encuesta diseñada a tal fin. Por último, analizaremos los resultados académicos obtenidos por los alumnos.

4.1. Metodología

Durante el curso tenemos dos tipos de sesiones: de teoría y de laboratorio. Las sesiones de teoría se imparten en aulas convencionales y las de laboratorio en aulas con ordenadores.

Las sesiones de teoría empiezan con el primer notebook de teoría, para introducir el lenguaje y el entorno de trabajo. A partir del segundo notebook de teoría, los alumnos trabajarán con ellos de forma autónoma en sus horas de estudio. Desde ese momento, las clases de teoría se convierten en talleres en los que (de forma guiada) se van resolviendo notebooks de ejercicios que contienen proyectos que permiten poner en práctica de forma progresiva distintos elementos del lenguaje.

Las clases de laboratorio comienzan un par de semanas después de las de teoría. Al igual que las clases de teoría también están centradas en proyectos. La diferencia está en que se trabaja con un entorno distinto (el IDE Eclipse) y que las sesiones son menos guiadas.

En lo tocante a la evaluación, seguimos un sistema de evaluación continua que ayuda a que los alumnos puedan avanzar de forma progresiva en la asignatura. El sistema contempla las siguientes pruebas:

- Cuatro cuestionarios de teoría: son pruebas en formato *test*, valoradas con 1 punto cada una, que se realizan a lo largo del cuatrimestre. Cada cuestionario se corresponde con un bloque de dos notebooks de teoría, de manera que los alumnos se van evaluando progresivamente sobre distintos aspectos de la programación en Python. Para cada uno de los cuatro bloques los alumnos tenían disponibles cuestionarios de autoevaluación (construidos sobre un total de 154 preguntas) que les han servido para medir su grado de conocimiento antes de enfrentarse a las pruebas.
- Un ejercicio de laboratorio: una prueba de laboratorio valorada con 6 puntos en la que los alumnos deben desarrollar un pequeño proyecto. La función de esta prueba es la de evaluar la capacidad del alumno para construir una solución a partir de un determinado problema que implique el tratamiento automático de información.

Los alumnos que no hayan superado la asignatura mediante la evaluación continua (sumar 5 puntos entre todas las pruebas anteriores) tienen la opción de un examen final consistente en un ejercicio de laboratorio, esta vez valorado con 10 puntos.

Un último punto en el plano metodológico, y que ha sido una sorpresa positiva para nosotros, es el aspecto de la organización entre los profesores de la asignatura. El cambio de enfoque y el nuevo material preparado han supuesto, como efecto secundario, una reducción considerable en el tiempo dedicado a la coordinación entre profesores. Durante el cuatrimestre se han coordinado 6 grupos de teoría y 15 de laboratorio, en los que han participado un total de 9 profesores distintos. La reducción del tiempo de coordinación ha venido de

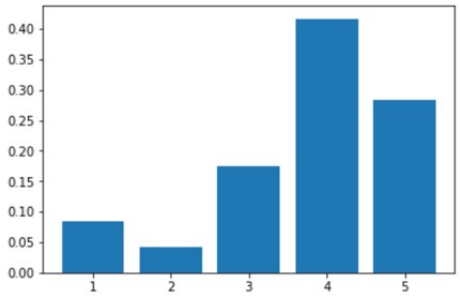

Figura 3: Opinión sobre la asignatura

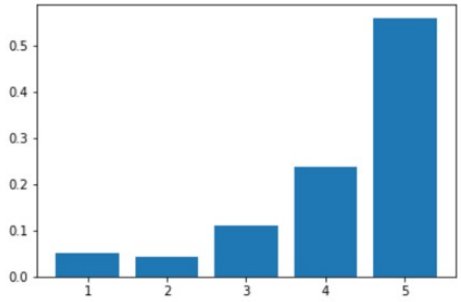

Figura 4: Opinión sobre el lenguaje Python

la mano de la estructura del material utilizado y del enfoque orientado a proyectos. En anteriores ediciones de la asignatura los mayores esfuerzos de coordinación estaban encaminados a acompasar el ritmo en el que se presentaban los contenidos, fundamentalmente para que no hubiese desajustes entre las clases de teoría y las de laboratorio. Con el nuevo diseño, la responsabilidad de la toma de contacto con el lenguaje cae de parte del alumno con su trabajo autónomo con los notebooks de teoría. Eso permite que tanto las clases de teoría como las de laboratorio se puedan centrar en el desarrollo y explicación de proyectos que, y esto es lo que provoca la no necesidad de coordinación, cada profesor puede elegir libremente y explicar al ritmo que considere oportuno. En definitiva, es un sistema que implica coordinación (gracias a los notebooks de teoría) y flexibilidad (gracias al enfoque orientado a proyectos).

4.2. Encuesta

Al final del cuatrimestre realizamos una encuesta anónima entre los alumnos para recabar sus opiniones sobre distintos aspectos de la asignatura, y en especial de aquellos relacionados con el cambio de enfoque que presentamos en este trabajo. La encuesta estaba compuesta por un total de 19 preguntas, de las cuales hemos seleccionado las más relacionadas con los elementos de nuestra experiencia docente para incluir un breve análisis en este trabajo.

La primera pregunta que analizaremos tiene que ver con el perfil de los alumnos (repetidor o no repetidor) y la opinión del cambio de planteamiento con respecto a cursos anteriores. En concreto, la pregunta fue:

- ¿En qué situación de las siguientes te encuentras? (No soy repetidor. Soy repetidor y la asignatura me ha gustado más que la del año pasado. Soy repetidor y la asignatura me ha gustado menos que la del año pasado.)

La figura 2 resume los resultados de esta primera pregunta. El diagrama de tarta muestra claramente que, entre los repetidores, la percepción del cambio es muy positiva. Para nosotros esto es un dato muy significativo ya que abre la puerta a la recuperación de alumnos que, por una razón u otra, abandonaron la asignatura en cursos anteriores. Este aspecto es especialmente importante en una asignatura de primero ya que puede influir de forma muy determinante en reducir la tasa de abandono de los títulos [10], uno de los principales indicadores en los sistemas de garantía de calidad de las titulaciones.

La figura 3 muestra los resultados de la siguiente pregunta que analizaremos, relativa a la opinión del método docente usado:

- Valora de 1 (poco adecuado) a 5 (muy adecuado) el método docente que hemos usado en la asignatura para enseñar a programar

El diagrama de barras muestra la distribución de valoraciones del método docente. Casi el 70 % de los alumnos lo valora con un 4 ó más, y la valoración media es de 3.775.

Por último, en la gráfica 4 se muestra la opinión sobre el lenguaje de programación usado. La pregunta que se formuló fue:

- Valora de 1 (poco adecuado) a 5 (muy adecuado) el lenguaje Python como primer lenguaje para aprender a programar

Los resultados muestran que a los alumnos, en general, les ha parecido que Python es un muy buen lenguaje para introducirse en la programación. Más del 50 % otorgaron la valoración máxima, y la valoración media fue 4.212.

4.3. Resultados académicos

El cuadro 4 muestra un resumen de los resultados académicos en los dos grados en los que se ha puesto en marcha nuestra experiencia, tanto en el curso 16/17 como en el 17/18. Durante el curso 16/17 el lenguaje de programación usado, las herramientas de desarrollo y la metodología docente fueron distintos. Sí fue común el sistema de evaluación, que en ambos cursos fue el mismo que se ha explicado en el apartado 4.1. Estas

	GISA		GII-TI	
	16/17	**17/18**	**16/17**	**17/18**
Matriculados	100	90	261	279
Presentados	56,0 %	76,6 %	63,2 %	70,6 %
Aprobados	38,0 %	68,8 %	29,1 %	43,0 %

Cuadro 4: Resultados en los cursos 16/17 y 17/18

dos circunstancias nos permiten usar esta comparativa para evaluar el impacto que han tenido los nuevos elementos introducidos en el curso 17/18.

En lo relativo al porcentaje de aprobados, los alumnos han obtenido mejores calificaciones que en el curso anterior, con una mejora del 14 % en el grado GII-TI y del 30 % en el grado GISA. Más significativa nos parece, incluso, la mejora en el porcentaje de alumnos presentados, que en ambos casos se ha conseguido llevar por encima del 70 %. Este aspecto está conectado directamente con nuestro objetivo inicial, que era trabajar sobre la motivación de los alumnos. En este sentido, estamos convencidos de que tanto el lenguaje Python, como los *notebooks* y el enfoque orientado a proyectos han contribuido a que más alumnos llevasen la asignatura al día, lo que ha redundado en la mejora del porcentaje de presentados.

5. Conclusiones

Hemos presentado una experiencia docente llevada a cabo durante el primer cuatrimestre del curso 17/18 en una asignatura de introducción a la programación. Los principales objetivos de nuestro trabajo eran a) mejorar la motivación de los alumnos y b) abandonar un esquema de enseñanza de programación en el que el ritmo de aprendizaje viene marcado por la estructura sintáctica del lenguaje al estilo del típico índice de un manual de un lenguaje de programación.

Nuestra propuesta se apoya en tres elementos: la flexibilidad y legibilidad del lenguaje Python, el entorno de programación interactiva Jupyter y un enfoque orientado a proyectos, que son los que marcan el ritmo de aprendizaje. El material docente desarrollado está disponible públicamente en un repositorio GitHub.

Los números, tanto de los resultados académicos como los del análisis de una encuesta realizada a los alumnos, son bastante alentadores y nos animan a seguir en esta dirección. Los resultados académicos mejoran significativamente con respecto al curso anterior, y de la encuesta se desprende que a los alumnos el cambio les ha parecido positivo.

Más allá de los números, las sensaciones (que son menos tangibles pero, en ocasiones, más reveladoras) son estupendas. Para los profesores la experiencia ha sido muy reconfortante porque el *feedback* ha sido siempre más positivo que en cursos anteriores. Se podría decir, en este aspecto, que hemos disfrutado de una *enseñanza significativa*.

Referencias

[1] David P. Ausubel. *The psychology of meaningful verbal learning*. Grune Stratton, Oxford, England, 1963.

[2] Brigid Barron. Linda Darling-Hammond. *Teaching for Meaningful Learning: A Review of Research on Inquiry-Based and Cooperative Learning*. Book Excerpt, George Lucas Educational Foundation, 2008.

[3] Jacob L. Bishop. Matthew A. Verleger. *The flipped classroom: A survey of the research*. ASEE National Conference Proceedings, Atlanta, vol. 30 (9), pp. 1-18, 2013.

[4] Phyllis C. Blumenfeld, Elliot Soloway, Ronald W. Marx, Joseph S. Krajcik, Mark Guzdial, Annemarie Palincsar. *Motivating project-based learning: Sustaining the doing, supporting the learning*. Educational psychologist, vol. 26, pp. 369-398, 1991.

[5] Charles Dierbach. *Python as a first programming language*. Journal of Computing Sciences in Colleges, vol. 29 (6), pp. 153-154, 2014.

[6] Fernando Pérez, Brian Granger. *IPython: a system for interactive scientific computing*. Computing in Science and Engineering, 9 (3), 2007.

[7] Ignacio Pozo. *Aprendices y Maestros: La Nueva Cultura del Aprendizaje, Psicología y educación*. Alianza, 1996.

[8] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, F. Frederic, M. Bussonnier. *The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication*. AGU Fall Meeting Abstracts, 2014.

[9] Guido Van Rossum, Fred L. Drake. *The python language reference manual*. Network Theory Ltd., 2011.

[10] David Ruiz. Francisco Gómez. José L. Ruiz. *Análisis de la tasa de abandono en un Centro con varios Grados en Ingeniería Informática*. Actas de las XXIII JENUI. Cáceres, pp. 173-180, 2017.

[11] Sibel Somyürek. *An effective educational tool: construction kits for fun and meaningful learning*. International Journal of Technology and Design Education, vol. 25 (1), pp. 25–41, 2015.

[12] Omar I. Trejos. *Relaciones de aprendizaje significativo entre dos paradigmas de programación a partir de dos lenguajes de programación*. Tecnura, 2014, vol. 18 (41), 2014.