UNIVERSIDAD DE GRANADA

Trabajo de Fin de Máster

Máster en Profesorado de Enseñanza Secundaria Obligatoria y Bachillerato, Formación Profesional y Enseñanzas de Idiomas

Autocorrección interactiva para la enseñanza y aprendizaje de la programación

Autor
Ernesto Serrano Collado

Tutor
Zoraida Callejas Carrión

Escuela Internacional de Posgrado

—

Granada, June 5, 2019

Autocorrección interactiva para la enseñanza y aprendizaje de la programación

Ernesto Serrano Collado

Resumen

Palabras clave: *educación, enseñanza, aprendizaje, programación, integración continua, sistemas de control de versiones, software libre*

Una de las mejores formas de aprender programación es a través de la realización de ejercicios, pero la labor de corrección de dichos ejercicios es una tarea que consume mucho tiempo. El coste de la corrección manual de ejercicios tiene una correlación directa con el número de actividades que podemos realizar con nuestros alumnos ya que a la hora de planificarlas hay que tener en cuenta el tiempo que nos va a llevar corregir dichas actividades.

Con herramientas y/o metodologías que nos faciliten dicha corrección podríamos proponer muchos más ejercicios e incluso generar material de extensión para alumnos con altas capacidades. Los alumnos que tengan necesidades especiales de apoyo educativo podrían hacer uso de los resultados de la corrección para entender dónde fallan. Asimismo los propios alumnos pueden ayudarse mutuamente incentivando el trabajo en equipo.

Vamos a proponer una metodología de trabajo que mediante el uso de sistemas de integración continua permitirá la corrección automática de ejercicios de programación reduciendo así el tiempo de corrección manual. Además los alumnos podrán ver el resultado de dicha autocorrección sirviéndoles como sistema de apoyo a su aprendizaje.

Para ilustrar la metodología se van a desarrollar una serie de ejercicios típicos de programación en los lenguajes de programación incluidos en el currículo de algunas de las asignaturas de Educación Secundaria, Bachillerato y Formación Profesional relacionadas con la informática.

Interactive autocorrection for the teaching and learning of programming

Ernesto Serrano Collado

Extended abstract

Keywords: *education, teaching, learning, programming, continuous integration, version control systems, free software.*

One of the best ways to learn computer programming is through exercises but the work of correcting such exercises is a time-consuming task. The cost of the manual correction of exercises has a direct correlation with the number of activities that we can carry out with our students, since when planning them we have to take into account the time it will take us to correct these activities.

With tools and/or methodologies that facilitate this correction we could propose many more exercises and even generate extension material for students with high capacities. Students with special needs for educational support could use the results of the correction to understand where they fail. Students can also help each others encouraging teamwork.

We are going to propose a working methodology that through the use of continuous integration systems will allow the automatic correction of programming exercises, thus reducing manual correction time. In addition, the students will be able to see the result of this self-correction, scaffolding their learning.

We will develop some example exercises to show how our methodology works, these exercises will be developed in the most common programming languages that are included in the curriculum of various subjects related with the computer programming.

There are many studies (Benotti, Aloi, Bulgarelli, and Gomez, 2018) and blogs posts like "How GitHub Classroom and Travis CI improved students' grades", 2019 that demostrates how self-correction and automatic code review systems improve programming learning by allowing students to self-evaluate immediately without having to wait for the correction of the exercises. In this way the student can see the correction of the exercise when he is working on it.

With the classic methods of manual correction the student must wait for this correction, days or even weeks can pass until seeing where it has failed

and in many cases having lost the context of what the student was trying to solve. We apply previous research and knowledge (Rubio and González del Valle, 2018) to achieve the goal of building our custom auto-correction system.

Yo, **Ernesto Serrano Collado**, alumno de la titulación **Máster en Profesorado de Enseñanza Secundaria Obligatoria y Bachillerato, Formación Profesional y Enseñanzas de Idiomas** de la **Escuela Internacional de Posgrado de la Universidad de Granada**, declaro que el presente Trabajo de Fin de Máster es original, no habiéndose utilizado fuentes sin ser citadas debidamente. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la Normativa de Evaluación y de Calificación de los estudiantes de la Universidad de Granada de 20 de mayo de 2013, *esto conllevará automáticamente la calificación numérica de cero [...] independientemente del resto de las calificaciones que el estudiante hubiera obtenido. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagien.*

Asimismo, autorizo la ubicación de la siguiente copia de mi Trabajo de Fin de Máster (*Autocorrección interactiva para la enseñanza y aprendizaje de la programación*) en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Además, este mismo trabajo está publicado bajo la licencia **Creative Commons Attribution-ShareAlike 4.0 CC** dando permiso para copiarlo y redistribuirlo en cualquier medio o formato, también de adaptarlo de la forma que se quiera, pero todo esto siempre y cuando se reconozca la autoría y se distribuya con la misma licencia que el trabajo original. Todo el código fuente así como este documento en formato `LaTeX` se puede encontrar en los siguientes repositorios de `GitHub`: https://github.com/erseco/ugr_tfm_maes y https://github.com/erseco/ugr_tfm_maes_sample_exercises.

Y para que así conste firmo el presente documento.

Fdo: Ernesto Serrano Collado

Granada, a June 5, 2019

D.ª **Zoraida Callejas Carrión**, profesora del **Departamento de Lenguajes y Sistemas Informáticos** de la **Universidad de Granada**.

Informa:

Que el presente trabajo, titulado ***Autocorrección interactiva para la enseñanza y aprendizaje de la programación***, ha sido realizado bajo su supervisión por **Ernesto Serrano Collado**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a June 5, 2019.

La tutora:

Zoraida Callejas Carrión

Agradecimientos

A Georgia, que algún día será mejor ingeniera que su tío.

Al ZX Spectrum 128K +2A de mis hermanos, porque sin él no habría llegado hasta aquí.

A mi *seño* Zoraida, por haber sido mi primera profesora en la carrera, por ponerme mi primera matrícula y por cerrar el círculo dirigiendo este TFM.

Contents

Chapter 1

Introducción

La democratización de Internet ha hecho que hoy día, al igual que en otras materias, sea impensable concebir la enseñanza sin hacer uso de Internet, tanto para la búsqueda de información como para la comunicación profesor-alumno y es posible que en el futuro la totalidad de la enseñanza se imparta a través de la red usando "MOOCs" o plataformas similares.

Aunque a efectos legales la labor de un profesor tenga un horario establecido (posiblemente de 8.00 a 15.00 horas), uno no deja de ser profesor, de hecho la mayoría de los profesores ejercen la profesión 24 horas al día, 7 días a la semana durante los 365 días del año. Dicha dedicación, que es en gran medida vocacional nos puede llegar a consumir gran parte de nuestro tiempo y por eso tenemos que apoyarnos en metodologías que agilicen nuestro trabajo. Imaginemos poder permitir a nuestros alumnos experimentar con los ejercicios y aprender por si mismos en un contexto donde no se penaliza el error todo ello sin requerir nuestra intervención directa. Puede sonar un poco utópico pero como dijo Richard Branson: "If your dreams don't scare you, they are too small".

Durante mis años de estudiante conocí a profesores que se limitaban a repetir año tras año la misma teoría, con unos ejercicios copiados de algún sitio de Internet sin tan siquiera plantearse que sus futuros alumnos también tendrían acceso a Internet y sabrían buscar la fuente original de dichos ejercicios, pero también tuve profesores a los que les gustaba *cacharrear* con todo tipo de nuevas tecnologías para mejorar su labor docente y mantenerse actualizados. Estos últimos han sido los que me han motivado tanto a llegar a ser profesor como a intentar definir, dentro de mis posibilidades, una metodología para optimizar la enseñanza de la programación.

Motivación

Mi primer contacto con la informática fue a finales de los ochenta. Un buen día mi padre apareció en casa con una caja que contenía un rudimentario teclado negro con un arco-iris pintado. A ese teclado le podías introducir cintas de casete como las que servían para escuchar música y tras esperar lo que ahora nos parecería una eternidad podíamos empezar a aporrear el teclado para llevar a Phantomas (figura 1.1) de pantalla en pantalla mientras íbamos esquivando enemigos y activando las palancas necesarias para abrir y desvalijar la caja fuerte de una mansión en el planeta Earth-Gamma.

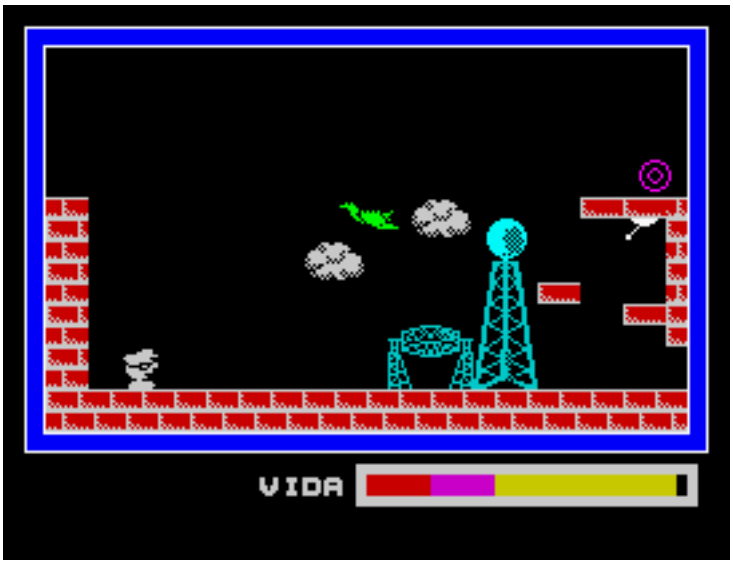

Figure 1.1: Phantomas (© 1986 Dinamic)

Todavía faltaba mucho tiempo para que aprendiera lo que era una interfaz (aún no había aprendido apenas a leer) pero ya sabía interpretar algunas letras, curiosamente las que tenían pintadas las teclas OPQA[1].

Según fui creciendo aprendí a leer y escribir y también a diferenciar entre texto y código, para mí el código eran un serie de letras que aparecían impresas en las últimas páginas de las revistas que compraba mi hermano. Esos códigos se llamaban POKES[2], y cuando los introducías en un juego conseguías cosas como vidas infinitas, la habilidad de atravesar paredes o la capacidad de ser invisible a los enemigos. Sin saberlo estaba aprendiendo a programar de forma muy rudimentaria.

[1] En los primeros ordenadores de la década de los 80 OPQA era la combinación de teclas que usaban la mayoría de los juegos siendo OP las teclas para desplazarse de izquierda a derecha y QA para saltar o moverse de arriba a abajo.

[2] Instrucción en lenguaje BASIC para guardar un valor en una dirección de memoria.

Algunos años más tarde aparecieron las consolas de videojuegos en las que metías un cartucho e instantáneamente estabas jugando a juegos increíbles con una paleta de colores que raro era que no provocara ataques epilépticos. Mientras mis amigos solo tenían que introducir el cartucho y empezar a jugar yo tenía que esperar 5 interminables minutos mientras escuchaba sonidos estridentes y cruzar los dedos por que no apareciera el fastidioso TAPE ERROR que se podía intentar solucionar girando un tornillo llamado "azimut" con un destornillador de estrella.

Y de esta manera, sin siquiera saberlo, tuve mis primeros contactos con el auto-aprendizaje, yo girando un tornillo sin saber a ciencia cierta el por qué y cuando el mayor problema que tenían mis amigos era que a veces tenían que dar un soplido fuerte al cartucho.

Pasaron los años, los sistemas se fueron haciendo más complejos y me di cuenta de que cuantas más opciones tenían los dispositivos, más perezosos se volvían los usuarios. Mi vecino sin ir más lejos, por no aprender a ajustar su televisor tenía "La 2" en el canal 3 ¿era yo el único que encontraba eso chirriante? Quizá no, pero como he ido descubriendo hay diferentes tipos de personas, están los que pueden pasar meses alumbrando el pasillo con el teléfono móvil y los que crean un tutorial en YouTube para enseñar a cambiar una bombilla.

Y ese es el deber que tenemos como futuros profesores, ser capaces de transmitir ese conocimiento e intentar que nuestros alumnos no pasen meses a oscuras, que aprendan a silenciar el volumen del altavoz del módem y en el mejor de los casos que sepamos despertar en ellos la curiosidad para que sean ellos mismos los que auto-aprendan y descubran todas esas cosas futuras que aun siendo profesores tenemos por aprender. *"In the future, the defining metric for organizations won't be ROI (Return on Investment), but ROL (Return on Learning)"* (Ismail, 2014).

Análisis general del problema

La programación es uno de los pilares fundamentales de la informática por lo que su aprendizaje es de carácter obligatorio. De hecho en muchos países como Finlandia, Alemania y Estonia promueven el aprendizaje de programación desde la educación primaria, en nuestro país *"el estudio de las CC en Educación Primaria y Secundaria se encuentra en su fase inicial y se ha empezado a introducir recientemente por lo que aún no ha sido adoptado por la mayoría de centros escolares. Como resultado, el número de niños y niñas que a día de hoy estudian ciencias de la computación es todavía una minoría"* (FECYT and Everis, 2016) pero se está promoviendo el aprendizaje a edades cada vez más tempranas.

La enseñanza de la programación suele considerarse bastante complicada, *"ya que muchos estudiantes encuentran bastantes dificultades cuando empiezan a programar"* (Rubio and González del Valle, 2018), de hecho *"la existencia de altas tasas de fracaso y la subsiguiente incapacidad de los estudiantes para escribir programas simples al final de una unidad de programación son solo dos de los problemas que inciden en las facultades de informática todo el mundo"* (Bruce, 2002).

Para enseñar programación a nuestros alumnos tenemos que tener en cuenta las dificultades que suponen la interacción ordenador/alumno y *"contar con una actitud receptiva a los problemas que les plantea a los alumnos la introducción de un lenguaje formal y su uso para programar y resolver problemas"* (Vitale, 1990).

La mejor manera de aprender a programar es, como en las matemáticas, haciendo muchos ejercicios. Hay artículos que afirman que la cantidad de ejercicios realizados por los estudiantes está directamente relacionado con la calificación obtenida (Qian, Hambrusch, Yadav, Gretter, and Li, 2019). Por ese motivo en casi todos los cursos de programación se proponen una gran cantidad de ejercicios adicionales. Evidentemente corregir todos estos ejercicios es una tarea bastante tediosa para los profesores por lo que habitualmente se opta por indicar una serie de ejercicios como obligatorios y el resto dejarlos como opcionales para que el alumno obtenga mejor calificación. La utilidad de realizar ejercicios se puede entender con la siguiente frase: *"when you move from point A to point B, you can then see point C. But you can't see point C from point A. Iteration/experimentation is the only way"* (Ismail, Palao, and Lapierre, 2018) o como diría Antonio Machado: "Caminante no hay camino, se hace camino al andar".

También vemos en la entrega de ejercicios un sistema de *caja negra* ya que el alumno no siempre sabe si el ejercicio que ha entregado está bien hecho y debe esperar a la corrección del mismo por parte del profesor para saber si lo hizo bien o no, si a eso le añadimos que el profesor puede tardar días o incluso semanas en entregar la corrección tenemos a un alumno que cuando vaya a comprobar la corrección habrá perdido el "contexto" de donde estaba, como se dice en el mundo de las metodologías ágiles es mucho más beneficioso conocer en que fallamos de manera temprana *"The sooner you realize that something is failing, the quicker you can fix it and the faster you will ultimately succeed. This would be the ultimate rule of thumb: When something is not working, change course. Fail Fast, Succeed Faster"* (Godse, 2013) .

El sistema habitual también nos impide sacar métricas de forma sencilla, con un sistema de corrección automática podemos saber el número de veces

que un alumno ha intentado realizar un ejercicios, cuanto ha tardado en dar con la solución correcta e incluso el tiempo que ha invertido en cada ejercicio. Todas estas métricas pueden ser muy útiles a la hora de calificar ya que a veces la programación es difícil de corregir y nos limitamos a una corrección "binaria", es decir, funciona o no funciona.

Además, la dinámica de mandar una serie de ejercicios y esperar a su entrega tiene el riesgo de que los alumnos plagien los ejercicios de otros alumnos del mismo curso o de cursos anteriores.

Estructura del proyecto

Antes de pasar a detalles más técnicos, me gustaría detallar el contenido de este proyecto:

- En el *capítulo 1* (**Introducción**) se encuentra una breve introducción a nuestra idea, así como las motivaciones que nos han llevado a realizarla.
- El *capítulo 2* (**Objetivos**) define los objetivos que se quieren alcanzar con este proyecto.
- En el *capítulo 3* (**Antecedentes**) se analiza el estado de arte actual, así como algunas de las tecnologías y paradigmas que utilizaremos en nuestro proyecto.
- En el *capítulo 4* (**Propuesta pedagógica**) se analizan los detalles pedagógicos del proyecto.
- En el *capítulo 5* (**Propuesta técnica**) se planifica cómo se van a realizar de forma técnica los contenidos del proyecto.
- En el *capítulo 6* (**Conclusiones**) se pueden encontrar las conclusiones finales así como las recomendaciones para futuros trabajos.

Para finalizar se incluye un anexo con el código fuente desarrollado y liberado bajo la licencia libre "GNU General Public License", 2007. Dicho código fuente también se puede encontrar en la url https://github.com/erseco/ugr_tfm_maes_sample_exercises/.

Chapter 2

Objetivos

El objetivo de este proyecto es definir una metodología de trabajo para enseñar programación haciendo uso de sistemas de control de versiones de código fuente que además, mediante un sistema software, permita la autocorrección de dichos ejercicios.

Dicho objetivo se descompone en los siguientes subobjetivos principales:

- **OBJ-1.** Analizar las posibilidades y el estado del arte actual de los sistemas de autocorrección actualmente existentes.
- **OBJ-2.** Determinar las ventaja que nos brindan los sistemas de autocorrección para mejorar la enseñanza y aprendizaje de la programación.
- **OBJ-3.** Definir una metodología de trabajo para la enseñanza de la programación haciendo uso de sistemas de control de versiones.

Además como objetivos secundarios tendremos:

- **OBJ-4.** Desarrollar una herramienta para la autocorrección de ejercicios de programación.
- **OBJ-5.** Escribir una serie de ejercicios de ejemplo que sirvan como guía de cómo utilizar nuestra metodología.
- **OBJ-6.** Definir las posibilidades adicionales de la metodología tanto para la enseñanza como para el aprendizaje de la programación.

Alcance de los objetivos

El fin inmediato de este informe es definir una metodología de trabajo para enseñar programación. Como objetivo adicional intentaremos desarrollar

una herramienta que permita a los profesores definir una serie de ejercicios que los propios alumnos puedan corregir de forma autónoma y automática de una manera sencilla.

Además todo el código así como la documentación resultante se ha liberado con una licencia libre para que cualquiera pueda hacer uso de la metodología, del código desarrollado así como de las conclusiones obtenidas.

Interdependencia de los objetivos

Todos los objetivos son interdependientes entre sí, pero el cuarto objetivo (**OBJ-3**) ha sido el principal motivador de este proyecto y es el que ha escudado y avalado el desarrollo de los demás. En aspectos más relacionados con la realización del proyecto, el tercer objetivo (**OBJ-3**) es el que nos ha brindado la base pedagógica sobre la que trabajar, ya que ha sentado la base sobre la que aplicar dicho cuarto objetivo (**OBJ-3**). El resto de objetivos secundarios, al no tener un carácter urgente han resueltos en base a la disponibilidad del tiempo necesario para su realización.

Conocimientos y herramientas utilizadas

Destacar en los aspectos formativos previos más utilizados para el desarrollo de esta metodología los conocimientos adquiridos en la asignatura `Procesos y contextos educativos` en todo lo referente a legislación y metodologías de enseñanza y la asignatura `Innovación docente e Investigación Educativa` por las ideas sobre cómo realizar una propuesta innovadora siendo ambas del **Máster en Profesorado de Enseñanza Secundaria Obligatoria y Bachillerato, Formación Profesional y Enseñanzas de Idiomas**.

También me gustaría destacar los conocimientos obtenidos en `Cloud Computing` para el análisis y configuración de los sistemas de integración continua y `Planificación y Gestión de Proyectos Informáticos.` para definir los requisitos y el planteamiento inicial del proyecto, ambas del **Máster Profesional en Ingeniería Informática** así como las asignaturas `Fundamentos de Programación` y `Programación Orientada a Objetos` del **Grado en Ingeniería Informática** para el desarrollo de los ejercicios de ejemplo.

Para la realización de cada una de las partes se han usado multitud de herramientas específicas tales como `LaTeX`, `Zotero`, `Travis CI` `Git` y `GitHub` entre otras.

Chapter 3

Antecedentes

En este capítulo vamos analizar el estado de arte actual y las tecnologías candidatas a utilizarse en este proyecto en base a los objetivos que presentamos en el capítulo anterior.

Procesos de verificación de código

Existen diferentes procesos para verificar el código, los más comunes son el desarrollo guiado por pruebas `TDD` y el desarrollo guiado por comportamiento `BDD`, ambos procedimientos tienen como finalidad verificar que el código desarrollado se adecua a los requisitos. De los dos el más sencillo de implementar es el guiado por pruebas ya que el basado en comportamiento requiere de mayor capacidad de abstracción. Por ello en nuestra metodología nos basaremos en TDD.

Aun así vamos a detallar en qué se basa cada uno para entender las diferencias de ambos procesos:

Desarrollo guiado por pruebas (TDD)

El desarrollo guiado por pruebas, en inglés `Test-Driven Development` y abreviado como **TDD**, es una práctica de programación que consiste en dos subtareas: primero escribir las pruebas, o *Test First Development* y después refactorizar, o *Refactoring*. Las pruebas son test unitarios, o *unit tests*. El procedimiento es sencillo, se escribe una prueba y se verifica que fallan ya que todavía no hemos escrito el código necesario para pasar la prueba. A continuación se escribe código que hace que la prueba se ejecute sin errores y por último se refactoriza el código escrito para mejorar su desempeño y/o legibilidad.

El propósito del desarrollo guiado por pruebas es lograr que nuestro código funcione y a la vez sea sencillo y legible. La idea es que los requi-

sitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que el software cumple con los requisitos que se han establecido. Como se indica en el estupendo libro Clean Code: *"Writing clean code is what you must do in order to call yourself a profesional. There is no reasonable excuse for doing anything less than your best"* (Martin, 2009).

Desarrollo guiado por comportamiento (BDD)

El desarrollo guiado por comportamiento, en inglés `Behavior-Driven Development` y abreviado como **BDD** es una práctica de programación que surgió a partir del TDD. Combina las técnicas generales y los principios del TDD junto con prácticas de análisis y diseño orientado a objetos para establecer herramientas que en teoría puedan utilizarse por equipos externos al de desarrollo para definir los test, sin implicar que ese equipo externo tenga conocimientos de programación.

A pesar de su definición teórica, la capacidad de abstracción necesaria para escribir buenos test basados en el comportamiento hace que en la mayoría de casos se necesiten conocimientos avanzados de programación.

Herramientas interactivas de corrección

Existen diversas herramientas de corrección de código que funcionan de forma interactiva permitiendo, vamos a enumerar las principales:

Coderunner

CodeRunner es un plugin gratuito de código abierto para Moodle que permite ejecutar el código fuente escrito por los estudiantes en respuesta preguntas de programación en muchos lenguajes diferentes. Está pensado principalmente para su uso en cursos de programación informática aunque puede utilizarse para calificar cualquier pregunta cuya respuesta sea texto. Su funcionamiento es tal que a una pregunta concreta sobre programación el estudiante deben escribir el código fuente en respuesta. Tras ello el código pasa a ejecutarse y pueden ver los resultados inmediatamente. Los estudiantes pueden corregir su código y volver a enviarlo, normalmente a cambio de una pequeña penalización.

Coderunner se puede obtener desde el sitio https://coderunner.org.nz

Python Tutor

Python Tutor es una herramienta de visualización de programas basada en web para Python, ya que dicho lenguaje se está convirtiendo en un lenguaje popular para la enseñanza de cursos introductorios de programación. Con

esta herramienta, los profesores y los alumnos pueden escribir programas de Python directamente en el navegador web (sin instalar ningún complemento), avanzar y retroceder a través de la ejecución para ver el estado de las estructuras de datos en tiempo de ejecución y compartir sus visualizaciones de programas en la web.

"En los últimos tres años, más de 200,000 personas han usado Python Tutor para visualizar sus programas" (Guo, 2013). Además, los instructores de más de una docena de universidades como UC Berkeley, MIT, la Universidad de Washington y la Universidad de Waterloo lo han utilizado en sus cursos de ciencias de la computación.

Python Tutor es un software gratuito y de código abierto, se puede obtener desde http://pythontutor.com.

Jupyter Notebooks

Jupyter Notebook (anteriormente IPython Notebooks) es un entorno basado en la web para la creación de documentos interactivos. El término "cuaderno" puede hacer referencia coloquialmente a muchas entidades diferentes, principalmente la aplicación web Jupyter, el servidor web Jupyter o el formato de documento Jupyter dependiendo del contexto. Un documento Jupyter Notebook es un documento JSON, siguiendo un esquema versionado, y conteniendo una lista ordenada de celdas de entrada/salida que pueden contener código, texto (usando Markdown), matemáticas, gráficos y medios enriquecidos, normalmente terminando con la extensión `.ipynb`. El proyecto Jupyter está licenciado bajo la BSD por lo que uso y distribución es gratuito, se puede obtener desde https://jupyter.org

Turingscraft's CodeLab

Esta herramienta comercial es la única de la lista que no tiene una licencia libre, está desarrollada por la empresa Neoyorquina Turingcrafts y desde 2002 ha sido utilizada por más de trescientos mil estudiantes de 20 países distintos (Barr and Trytten, 2016).

Se puede obtener más información desde su web http://www.turingscraft.com/

Chapter 4

Propuesta Pedagógica

En este capítulo pasamos a detallar la ventaja pedagógica que se pretende obtener en en base a los objetivos que presentamos en el segundo capítulo y a los antecedentes vistos en el tercero.

Análisis de las soluciones

Nos gustaría proponer una metodología donde los alumnos usen herramientas que van a tener que usar en futuro profesional (o académico) como pueden ser los sistemas de control de versiones y los sistemas de integración continua, o *continuous integration (CI)*, hay que tener en cuenta que esta metodología está enfocada mayoritariamente a la Formación Profesional porque es la que más asignaturas relacionadas con la programación, como veremos más adelante. Como la finalidad Formación Profesional es enseñar una profesión enseñarles a nuestros alumnos a usar estos sistemas les va a ser de gran ayuda a la hora de conseguir empleo.

Las tres soluciones libres descritas en los antecedentes (Coderunner, Python Tutor y Jupyter Notebooks) obligan a nuestros alumnos a aprender a usar una herramienta concreta que posiblemente no vayan a volver a utilizar jamás. Esto sería comparable a lo que didácticamente decimos de que el alumno "aprende a aprobar exámenes" mas que aprender los contenidos de la asignatura.

Por eso hemos optado por utilizar dos de las tecnologías más utilizadas para gestión de código fuente e integración continua como pueden ser GitHub y Travis CI. Además, ambas son de uso gratuito para proyectos de software libre, fomentando así su uso como se indica en la Orden de 21 de febrero de 2005, sobre disponibilidad pública de los programas informáticos de la Administración de la Junta de Andalucía y de sus Organismos Autónomos.

Temario relacionado con la programación informática

Tal y como indica el Decreto 110/2016 de 14 de junio, por el que se establece la ordenación y el currículo del Bachillerato en la Comunidad Autónoma de Andalucía las tecnologías de la información y de la comunicación para el aprendizaje y el conocimiento se utilizarán de manera habitual como herramientas integradas para el desarrollo del currículo. Asimismo en los currículos de las distintas ramas educativos donde podemos ejercer como profesores incluyen asignaturas relacionadas con la informática y de forma más específica con la programación informática.

"Tecnologías de la Información y Comunicación es un término amplio que enfatiza la integración de la informática y las telecomunicaciones, y de sus componentes hardware y software, con el objetivo de garantizar a los usuarios el acceso, almacenamiento, transmisión y manipulación de información. Su adopción y generalización han provocado profundos cambios en todos los ámbitos de nuestra vida, incluyendo la educación, la sanidad, la democracia, la cultura y la economía, posibilitando la transformación de la Sociedad Industrial en la Sociedad del Conocimiento." de Andalucía, 2016

La oferta de asignaturas de informática es variopinta, pues en la Educación Secundaria y Bachillerato es muy básica pero muy específica en la Formación Profesional, por lo que es complejo tener una metodología que se adapte a todos los niveles. Asimismo, los temarios suelen ser bastante extensos incluyendo tanto contenido que es muy difícil abordar cada tema en profundidad. Aquí podemos ver una de las ventajas de nuestra propuesta ya que por un lado, el tiempo que ahorramos lo podemos invertir en impartir más contenidos y que podemos centrarnos en las partes donde fallan los alumnos al ver el histórico de sus errores.

A continuación vamos a enumerar las distintas asignaturas relacionadas con la informática que se pueden encontrar en los currículos de Educación Secundaria, Bachillerato y Formación Profesional, además indicaremos cuales son candidatos a utilizar nuestra metodología en base a sus contenidos.

Educación Secundaria Obligatoria (ESO)

En la Orden de 14 de julio de 2016 se especifica el currículo de la materia de "Tecnologías de la Información y Comunicación" que es una materia de opción del bloque de asignaturas específicas para el alumnado de cuarto curso de la Educación Secundaria Obligatoria. Es una asignatura de carácter optativo que intenta dar una visión global de la informática, con lo que su temario intenta abarcar desde historia de la informática hasta la seguridad o la programación web pasando por la ofimática y estudio de los elementos de hardware, nuestra metodología propuesta podría servir de ayuda para la

enseñanza de programación de páginas web con HTML que es uno de los contenidos que se dan en dicha asignatura.

Bachillerato

En la Orden de 14 de julio de 2016, por la que se desarrolla el currículo correspondiente al Bachillerato en la Comunidad Autónoma de Andalucía indica las diferentes modalidades de bachillerato, incluyendo las modalidades de Ciencias y la de Humanidades y Ciencias Sociales las asignaturas de carácter obligatorio "Tecnologías de la Información y de la Comunicación I" y "Tecnologías de la Información y de la Comunicación II" y habiendo también una asignatura de libre configuración autonómica llamada "Programación y Computación". Estas asignaturas contienen contenidos de programación informática y de programación de páginas web por lo tanto son susceptibles de utilizar nuestra metodología de realización de ejercicios.

Ciclo Formativo Grado Medio (CFGM)

La Orden EDU/2187/2009, de. 3 de Julio establece el currículo del ciclo formativo de Grado Medio correspondiente al título de "Técnico en Sistemas Microinformáticos y Redes (SMR)" que es el único ciclo de grado medio relacionado con la informática donde la única asignatura ligeramente relacionada con la programación es "Aplicaciones web", pero dicha asignatura se centra más en el despliegue de aplicaciones que en la programación en sí, por lo que aunque nuestra metodología se podría adaptar para enseñar a configurar como desplegar aplicaciones web no entra dentro del alcance de este trabajo por lo que se deja como posible trabajo futuro.

Ciclo Formativo Grado Superior (CFGS)

Existen tres ciclos formativos de grado superior del plan LOE relacionados con la informática, la Orden EDU/392/2010, de 20 de enero, por la que se establece el currículo del ciclo formativo de Grado Superior correspondiente al título de "Técnico Superior en Administración de Sistemas Informáticos en Red" describe las asignaturas "Lenguajes de Marca y Sistemas de Gestión de Información" y "Gestión de Bases de Datos" en las que nuestra metodología se puede aplicar para corregir los ejercicios ya que se enseñan los lenguajes HTML, SQL, Javascript, XML así como algunos subconjuntos de XML como son DTD y XSD. Los currículos del título de "Técnico Superior en Desarrollo de Aplicaciones Web" definido en la Orden EDU/2887/2010, de 2 de noviembre y el de "Técnico Superior en Desarrollo de Aplicaciones Multiplataforma" definido en la Orden EDU/2000/2010, de 13 de julio definen varias asignaturas, algunas comunes, como pueden ser "Programación", "Bases de Datos" y "Lenguajes de marcas y sistemas de gestión de información" y otras específicas de cada título como pueden

ser “Programación multimedia y dispositivos móviles” o “Desarrollo web en entorno cliente” que están centradas en el aprendizaje de la programación informática y en los que si encajaría nuestra metodología.

Formación Profesional Básica (FPB)

La Orden ECD/1030/2014, de 11 de junio define el “Título Profesional Básico en Informática y Comunicaciones” y la Orden ECD/1633/2014, de 11 de septiembre define el “Título Profesional Básico en Informática de Oficina”, siendo ambas titulaciones las únicas relacionadas con la informática de todas las que componen la formación profesional básica, pero al igual que con el ciclo formativo de Grado Medio ambos currículos carecen de temario específico de programación por lo que queda fuera del ámbito de este proyecto.

Encuesta de percepción

Se realizó una breve encuesta y se difundió entre algunos profesores de diferentes ámbitos con el fin de ver si nuestra hipótesis de que la corrección de ejercicios era la tarea más tediosa que realizaban como docentes.

Aprovechamos la encuesta para obtener también algunos datos interesantes sobre la forma de trabajar de los docentes, de cara a trabajos futuros. La encuesta se podía consultar en la siguiente dirección web: https://forms.gle/xxSz6mQDjswo5B137.

La encuesta estuvo activa desde el 19 hasta el 25 de Mayo de 2019 y se recopilaron un total de 47 respuestas. Es una población pequeña pero aun así es interesante por las características de la misma ya que la encuesta se compartió “de boca a oreja” indicando que sólo lo compartieran con otros profesores a ser posible relacionados con la informática para no desvirtuar las respuestas.

Por supuesto los resultado de la encuesta son orientativos y no tienen un fundamento estadístico debido a los medios y el tiempo con que se contaba, pero que nos ha servidor para tener una impresión general.

1. **¿Dónde sueles impartir tus clases?**

   Con esta pregunta queríamos tener una visión de la población encuestada, ya que al difundirla entre profesores y animarles a compartirla entre sus compañeros profesores era muy posible que sus red de contactos no se limitara a profesores de Educación Secundaria, así que dejamos elegir entre las siguientes opciones:

- Educación Primaria
- Educación Secundaria
- Universidad (Grado/Máster)
- Practicas MAES

La gran mayoría de los encuestados han resultado ser profesores de Educación Secundaria, por lo que los resultados han sido satisfactorios, los siguientes han sido estudiantes en prácticas en el Máster en Profesorado de Enseñanza Secundaria Obligatoria y Bachillerato, Formación Profesional y Enseñanzas de Idiomaslos cuales podríamos englobar dentro del conjunto de los de Secundaria, llegando así casi al 90% de la población encuestada. Le siguen la respuestas de algunos profesores de Universidad y por último algunos maestros de Educación Primaria.

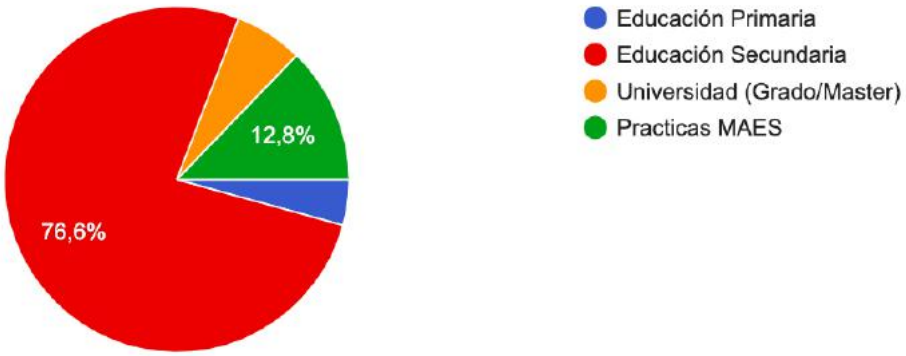

Figure 4.1: ¿Dónde sueles impartir tus clases?

2. **Valora las siguientes tareas que sueles realizar**

   Esta es quizá la pregunta más importante de toda la encuesta, porque queríamos saber la opinión de diversos profesores sobre las tareas de corrección de ejercicios y/o exámenes. Además nos ha servido para hacernos una idea de lo que opinan de las tareas más comunes de un profesor, había que valorarlas según se consideraban “divertidas”, “normales” o “tediosas” entendiéndose tediosas como aburridas. Las opciones eran las siguientes:

   - Preparación de material
   - Preparación de clases
   - Impartir clases

- Corrección de ejercicios
- Evaluación final
- Tutorías

Como podemos ver, la hipótesis de que corregir ejercicios resulta tedioso ha sido validada. De hecho ha sido la única de las opciones que no ha recibido ningún voto como actividad "divertida". Del resto de preguntas podemos deducir que a la mayoría de profesores les divierte impartir clases, esto es algo normal. No tendría sentido una respuesta diferente dado el carácter vocacional de la profesión, y si nos centráramos en los aspectos económicos es más flagrante en el mundo de la informática porque los sueldos de la empresas privadas están muy por encima de lo que se puede llegar a cobrar como docente. El resto de preguntas se han considerado normales, habiendo opiniones en las tres vertientes. Quizá remarcar que la evaluación es la que menos votos de tarea "divertida" han recibido. Aunque esto puede variar mucho de un centro a otro.

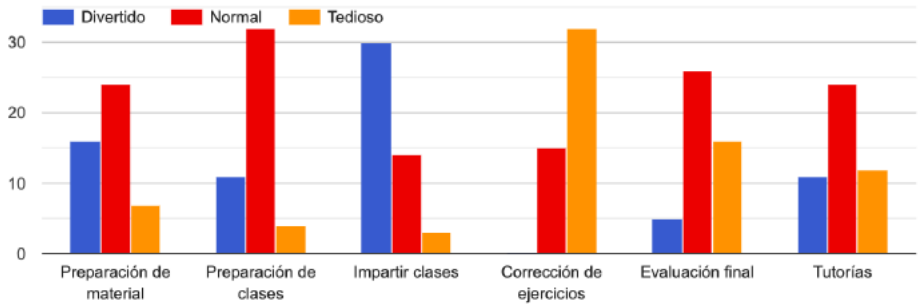

Figure 4.2: Valora las siguientes tareas que sueles realizar

3. **¿Elaboras tus propios ejercicios?**

   En esta pregunta queríamos saber si los profesores elaboran sus propios ejercicios. En mi experiencia como alumno así como en mi experiencia como docente en prácticas he visto exactamente lo que se ve reflejado en la encuesta, hay profesores que elaboran todo su material, los hay que lo hacen de forma regular pero no siempre, también los hay que los realizan de forma ocasional. Aquí me gustaría remarcar que la opción "Nunca" ha recibido muy pocos votos cuando en mi experiencia también hay docentes que utilizan el material de otros profesores en lugar del suyo propio.

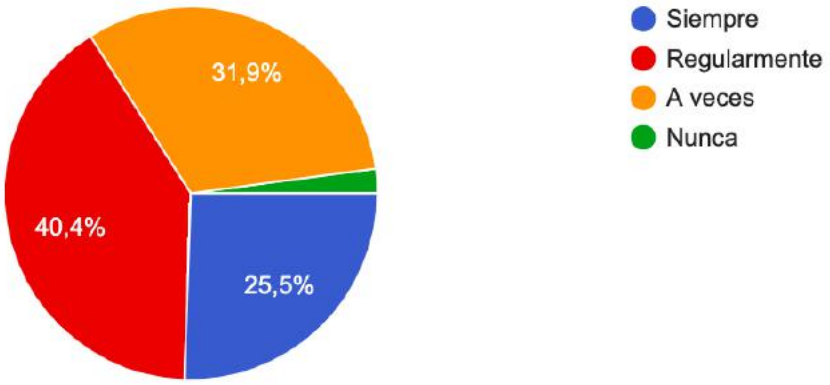

Figure 4.3: ¿Elaboras tus propios ejercicios?

4. **¿Crees que hay que compartir los ejercicios con el resto de la comunidad?**

   Esta pregunta era para ver la repercusión que tiene a nivel docente los conceptos del software libre y las licencias abiertas como la GPL y la "Creative Commons" Comons, 2013 así como la WikiPedia y otros proyectos que abogan por la difusión libre y gratuita tanto de software como de todo tipo de contenidos culturales.

   Por nuestra experiencia pensábamos que no iba a haber tan buena acogida del "Sí" como ha habido, pero nos ha sorprendido gratamente que la gran mayoría de los profesores encuestados apoyen el compartir el material. También es algo que se puede considerar normal ya que hoy en día es difícil hacer uso de recursos de terceros encontrados online para apoyar nuestras clases. De hecho en mi periodo como docente en prácticas me sirvió para ver que al contrario que en mi época de estudiante, donde usábamos el libro de texto, los docentes usan recursos como Kahoot![1] o YouTube para hacer las clases mucho más dinámicas.

[1]Plataforma gratuita que permite la creación de cuestionarios de evaluación en forma de concurso.

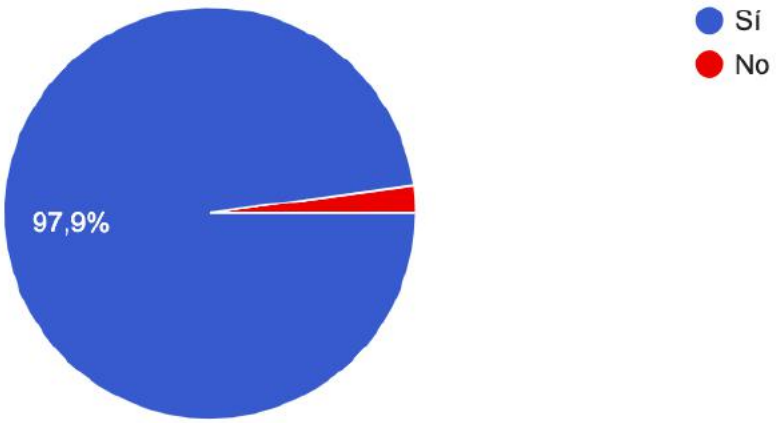

Figure 4.4: ¿Crees que hay que compartir los ejercicios con el resto de la comunidad?

5. **¿Sueles entregar correcciones de los ejercicios?**

   Esta pregunta está íntimamente relacionada con el propósito de este trabajo, queríamos saber si los profesores solían entregar los ejercicios corregidos, hay que tener en cuenta que para que dicha corrección tenga utilidad las correcciones se han de entregar de forma temprana, ya que es cuando el alumno aun está adquiriendo los conocimientos, de dieron las opciones “Sí”, “No” y “Otra”, siendo esta última opción para que los profesores indicaran alguna opción distinta, a continuación se pega en bruto dichas opciones alternativas:

   - Algunas veces
   - A veces y otras se corrigen en clase.
   - Sí o se corrigen en clase.

   Nos pareció interesante la opción de la corrección en clase, pues no la habíamos tenido en cuenta, pensamos que es una forma muy didáctica de que el alumno vea dónde ha fallado y hacer una corrección temprana, pero teniendo en cuenta que por norma general siempre hay una sensación de falta de tiempo para impartir el temario, así que quizá invertir tiempo en esta corrección puede hacer que lo perdamos para impartir nuevos contenidos. Esto no quiere decir que se siga invirtiendo tiempo en realizar ejercicios en clase, q

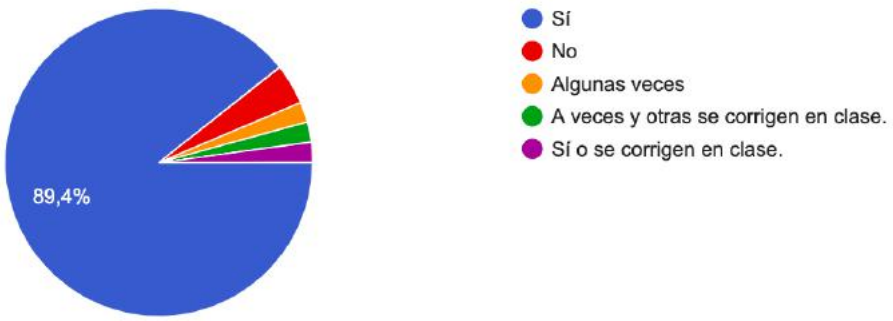

Figure 4.5: ¿Sueles entregar correcciones de los ejercicios?

6. **¿Qué opinas de los sistemas de corrección automática?**

   Aquí preguntamos de forma directa sobre los sistemas de corrección automática de forma genérica, sin especificar ninguno, ya que como hemos visto en los antecedentes existen diversos métodos, aquí les dimos a elegir entre estas opciones:

   - Ahorran tiempo a alumnos y profesores
   - Mejoran la enseñanza al poder ver de manera más inmediata los resultados
   - Desvirtúan la finalidad de los ejercicios propuestos
   - Deshumaniza la relación profesor-alumno
   - Otra (especificar)

   La opción "Mejoran la enseñanza al poder ver de manera más inmediata los resultados" fue marcada por casi el 50% de los encuestados, seguida por la de "Ahorran tiempo a alumnos y profesores" que consiguió algo más del 20%, ambas en conjunto eran las opciones que eran más acordes al desarrollo de nuestra metodología por lo que podemos ver que hay un claro interés en estos sistemas, además pusimos la opción "Otra" donde los profesores indicaron opiniones adicionales que pasamos a enumerar:

   - Ahorran tiempo a todos y mejoran la enseñanza. Se pueden hacer más ejercicios y practicar mucho es esencial en enseñanzas técnicas.
   - Sirven para ver si los alumnos han adquirido los conocimientos.
   - Es otro tipo de herramienta de evaluación.
   - No lo he probado.

- Son útiles en su justa medida.

En las respuestas vemos que en su mayoría también coinciden que los sistemas de corrección automática son muy útiles para mejorar la enseñanza, así que creemos que nuestra propuesta puede ser de gran ayuda a profesores.

Figure 4.6: ¿Qué opinas de los sistemas de corrección automática?

7. **Sugerencias para mejorar las clases**

   Incorporamos a la encuesta un apartado para que los profesores aportaran sus sugerencias para mejorar las clases, a continuación vamos a poner los datos recopilados en bruto en dicho apartado:

   - Hacerlas dinámicas y participativas, huir de la explicación meramente.
   - Lo más eficiente para el aprendizaje del alumnado es que ellos mismos corrijan, así si se les indican los errores suelen verlos y evitarlos después.
   - Que parte de la evaluación la hagan los propios alumnos (son más conscientes de los errores cometidos).
   - Fomentar el trabajo basado en proyectos colaborativos.
   - Correcciones en clase de la totalidad de los ejercicios, haciendo partícipes a los alumnos.
   - Coevaluación, aula invertida. Cooperativo.
   - Participación continuada de alumno.
   - La tipología de ejercicio más eficiente a la hora de su corrección es el tipo test. Existe una optimización de tiempo de corrección bastante notable en comparación con ejercicios con preguntas de

desarrollo, donde se tarda más por no tener tan automatizada la respuesta correcta (hay que leer la respuesta), la puntuación se somete a matices por estar incompleta, en ocasiones es difícil comprender la letra del alumno, etc.

Definición de la metodología

Tras validar que nuestra propuesta efectivamente tiene un alto valor pedagógico y que encaja dentro de los contenidos de varias asignaturas de los currículos de educación secundaria pasamos a definir los objetivos, contenidos y competencias de nuestra metodología.

El objetivo de nuestra metodología es enseñar programación de una forma diferente, con ejercicios, autocorrección y con tecnologías que se usan en el día a día de cualquier informático que se dedique profesionalmente a ello. La misma nos brinda muchas ventajas como puede ser ver un histórico de los envíos de un alumno, para saber donde falla o el tiempo que le ha dedicado a un ejercicio concreto, un simple vistazo al "Travis CI" de cada alumno nos da mucha información, y el poder seguir la traza es muy útil, también podemos ver el historial de *commits* de *Git* de forma rápida para ver la evolución del alumno. El código de colores de *Travis CI* también hace que corregir sea intuitivo, verde para correcto y rojo para incorrecto. En un momento dado podríamos detectar un posible plagio entre alumnos de forma rápida usando el mencionado historial.

Como indica el Decreto 110/2016 de 14 de junio el alumno debe *"conocer el funcionamiento de las nuevas tecnologías de la información y la comunicación, comprendiendo sus fundamentos y utilizándolas para el tratamiento de la información (buscar, almacenar, organizar, manipular, recuperar, presentar, publicar y compartir), así como para la elaboración de programas que resuelvan problemas tecnológicos"* (de Andalucía, 2016), así que además de todo lo indicado anteriormente estamos ajustándonos a la normativa.

El decreto anteriormente citado dice que "la oferta curricular diseñada en este Decreto potencia el desarrollo de las tecnologías de la información y la comunicación, así como la enseñanza de las lenguas extranjeras, teniendo en cuenta los objetivos emanados de la Unión Europea en esta materia y los planes estratégicos de la Comunidad Autónoma de Andalucía para el desarrollo de las lenguas" por lo que todos los ejercicios propuestos han sido realizados en inglés ya que es el idioma predominante en la informática y nuestros alumnos deberán adquirir competencias de lectura y comprensión en lengua inglesa para poder aprender por si mismos, de hecho el comando *Git* y las páginas *GitHub*, *Travis CI* están completamente en inglés así como la mayor parte de la documentación de lenguajes de programación. Sin ir

más lejos la propia programación informática define sus directivas en inglés, con palabras como “function”, “class” “for”, “if”, “then” o “while”.

Contenidos

Los contenidos de nuestra metodología comprenderán dos partes. Una parte común a todas las asignaturas donde se enseñará a usar Git y algunos conceptos básicos de programación y otra específica donde haciendo uso de lo aprendido en la parte común se seguirá el temario de la asignatura en cuestión.

La parte común comprenderá los siguientes dos apartados que se definirán de modo más extenso en la propuesta técnica:

- Aprendiendo a usar Git: Comandos básicos, creación de una cuenta en GitHub, integración continua, realizar un *fork* del repositorio de ejercicios ...
- Introducción a la programación: Conceptos comunes a todos los lenguajes como “variable” y “constante”, condicionales y bucles,...

Esta parte común podrá resumirse o incluso obviarse si los alumnos ya conocen alguno de los dos conceptos básicos por haberlos aprendido en otra asignatura.

La parte específica se desarrollará en base a los contenidos de la asignatura a impartir, pero en la propuesta técnica se definirán algunos ejercicios de ejemplo de diferentes asignaturas como pueden ser “Lenguajes de marcas y sistemas de gestión de información”, “Programación” o “Bases de Datos”.

Los ejercicios de la parte específica estarán siempre disponibles en su totalidad para que los alumnos puedan ir avanzando al ritmo que deseen. Cada ejercicio tendrá una breve explicación de lo que se requiere y una serie de test TDD que deberán pasar. El profesor también podrá poner ejercicios resueltos pero con algunos errores para que los alumnos los corrijan.

Competencias

La programación informática, y por extensión nuestra metodología, permite adquirir las siguientes 5 del total de 7 competencias clave de la LOMCE descritas en la Orden ECD/65/2015, de 21 de enero:

- **CCL**: Competencia en comunicación lingüística
- **CMCT**: Competencia matemática y competencias básicas en ciencia y tecnología

- **CD**: Competencia Digital
- **CPAA**: Competencia de aprender a aprender
- **SIE**: Sentido de la iniciativa y espíritu emprendedor

Entrando en detalle, de nuestra metodología, podemos destacar y definir las siguientes tres:

- Utilizar las tecnologías de la información y la comunicación de modo habitual en el proceso de aprendizaje, buscando, analizando y seleccionando información relevante en Internet o en otras fuentes, elaborando documentos propios, haciendo exposiciones y argumentaciones de los mismos y compartiendo éstos en entornos apropiados para facilitar la interacción.
- Conocer el funcionamiento de las nuevas tecnologías de la información y la comunicación, comprendiendo sus fundamentos y utilizándolas para el tratamiento de la información (buscar, almacenar, organizar, manipular, recuperar, presentar, publicar y compartir), así como para la elaboración de programas que resuelvan problemas tecnológicos.
- Asumir de forma crítica y activa el avance y la aparición de nuevas tecnologías, incorporándolas al quehacer cotidiano.

Proceso de entrega

Para la entrega de ejercicios el alumno deberá realizar un "Pull request" contra el repositorio del profesor lo cual dará el ejercicio por entregado. El repositorio tiene una plantilla para la creación de dichos *pull requests* en la que al alumno comprobará un checklist indicando que ha leído el guión de la práctica, cumple los posible pre-requisitos (por ejemplo haber terminado un ejercicio previo) así como indicar que el código entregado es original y no ha tratado de hacer modificaciones en el TDD ni en el *pipeline* (figura 4.7).

Al crear el *pull request* el alumno deberá verificar que no hay conflictos con el código base, si los hubiera deberá corregirlos manualmente. Tras esto se lanzarán los *pipelines* del repositorio del alumno (se activan en cada envío) y el del repositorio del profesor, que se activa al crear el *pull request* (figura 4.8) y al hacer cualquier envío adicional a la rama asociada al *pull request*. Tras terminar el proceso de ejecución del *pipeline* veremos si este ha dado un error (figura 4.9) o ha terminado de forma satisfactoria (figura 4.10) siendo esta última la imprescindible para proceder a corregir el ejercicio (figura 4.11).

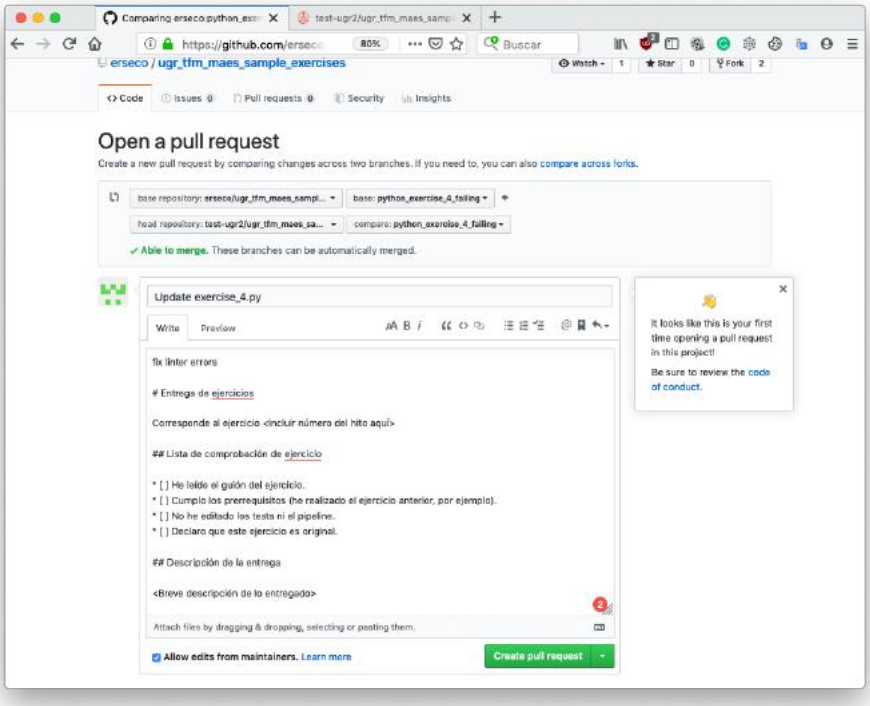

Figure 4.7: Plantilla de creación de Pull Request

Evaluación

Para la evaluación de los ejercicios nos basaremos en primer lugar en si han seguido los pasos indicados de forma correcta, realizando un "Pull request" e indicando lo realizado, luego comprobaremos si los "pipelines" han pasado de forma correcta y de forma adicional podemos entrar a valorar, como hemos mencionado anteriormente, el histórico de envíos a GitHub para ver si ha ido solucionando los problemas de forma adecuada y también valorando el tiempo dedicado. Esto no quiere decir que a más tiempo mayor calificación, pero si que sirve para hacernos una idea del esfuerzo que ha invertido el alumno para solucionar los ejercicios. Esto no quiere decir que también se pueda realizar una corrección "manual" de los ejercicios si fuera necesario.

Una característica muy útil de nuestro sistema es que podemos ver los cambios de forma visual con la pagina de diferencias, o *diff* (figura 4.12) lo que nos permite centrarnos en los cambios que ha realizado el alumno, además de detectar cualquier posible intento de modificación del código del *pipeline* o de los tests de verificación, con esto nos podemos asegurar en la evaluación de que todo lo entregado por el alumno está realizado conforme

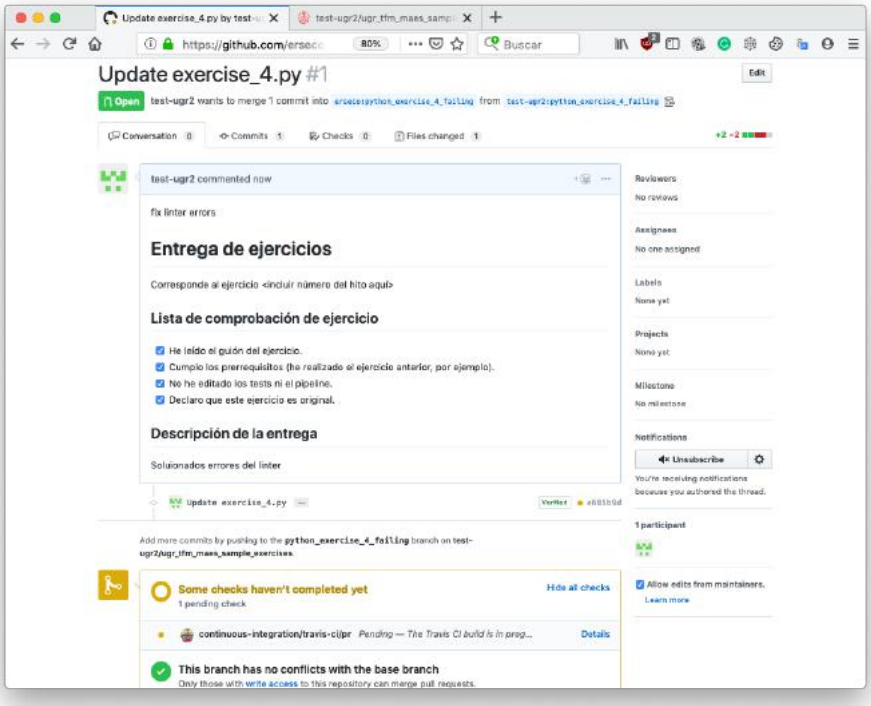

Figure 4.8: Vista de un Pull Request creado con el pipeline en ejecución

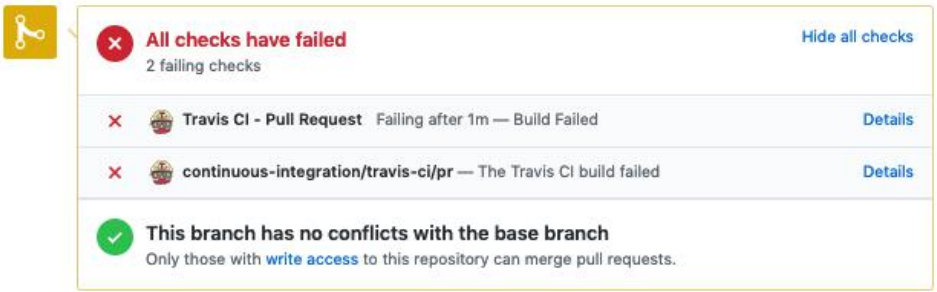

Figure 4.9: Detalle de un pipeline fallido

al guión entregado.

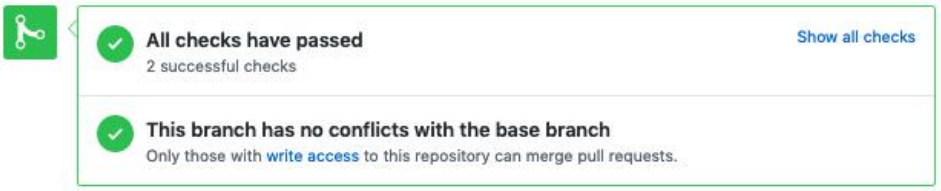

Figure 4.10: Detalle de un pipeline satisfactorio

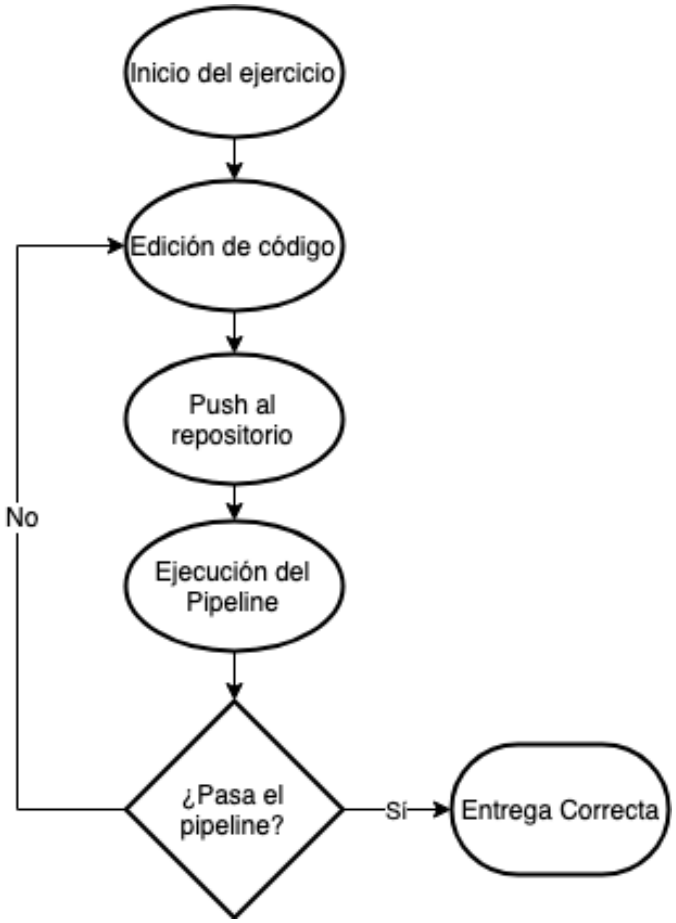

Figure 4.11: Diagrama de flujo de entrega de un ejercicio

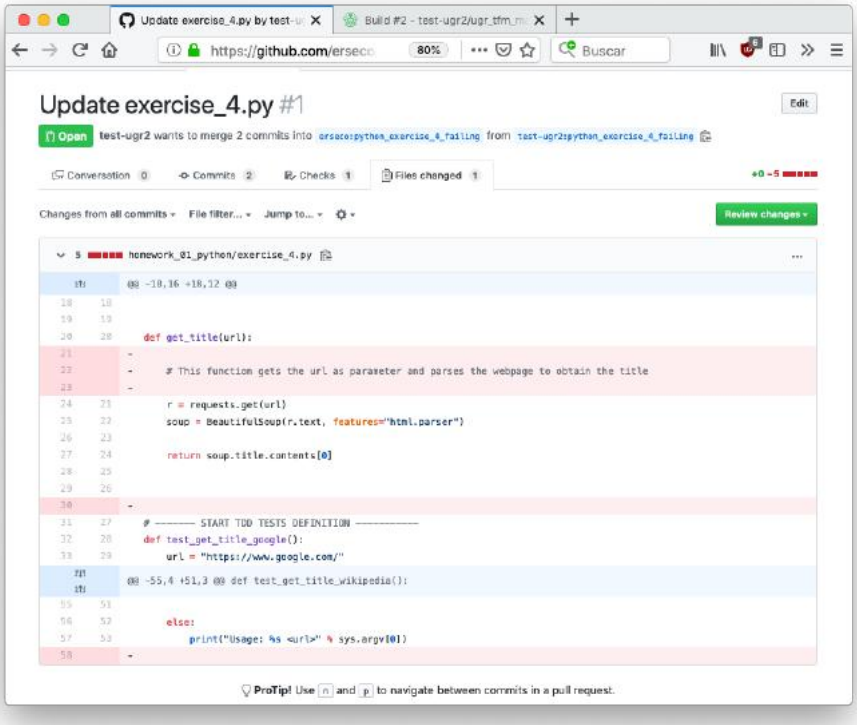

Figure 4.12: Detalle de los cambios de código en un Pull Request

Chapter 5

Propuesta Técnica

En este capítulo vamos a definir y a detallar los procesos que vamos a seguir para llevar a cabo la propuesta pedagógica definida en el capítulo anterior.

Herramientas necesarias

Sistemas de control de versiones

Lo primero que necesitaremos es enseñar a nuestros alumnos a utilizar un sistema de control de versiones, aunque existen varios sistemas (SVN, Mercurial, CVS), hoy en día estándar de facto es Git que además es software libre.

Para una administración más sencilla de nuestro código utilizaremos un cliente web como es GitHub, aunque existen otros como pueden ser GitLab o BitBucket.

Git

Git es un sistema de control de versiones distribuido que nos ayuda a llevar un seguimiento de los cambios en el código mientras desarrollamos. Se diseñó para su uso con código fuente pero se puede utilizar para llevar el seguimiento de los cambios en cualquier conjunto de archivos. Sus características incluyen la velocidad, la integridad de los datos y la compatibilidad con flujos de trabajo distribuidos y no lineales.

Git fue creado por Linus Torvalds en 2005 para el desarrollo del núcleo de Linux como alternativa al software BitKeeper que era un sistema de control de versiones propietario tras cambios en la licencia de este último.

Como con la mayoría de los sistemas de control de versiones distribuidos, y a diferencia de la mayoría de los sistemas cliente-servidor cada directorio de Git es un repositorio completo con un historial completo y capacidades de seguimiento de versiones completas, independiente del acceso a la red o a un servidor central.

Git es software libre y de código abierto distribuido bajo los términos de la Licencia Pública General GNU versión 2.

GitHub

GitHub es un servicio de alojamiento basado en web para el control de versiones a través de Git, fue adquirido por MicroSoft en 2018. Se utiliza sobre todo para código fuente aunque se puede utilizar para almacenar todo tipo de contenidos, incluso libros. Ofrece toda la funcionalidad de control de versiones distribuido y gestión de código fuente de Git, además de añadir sus propias características.

Proporciona control de acceso y varias funciones de colaboración como seguimiento de errores, *pull requests*, gestión de tareas y *wikis* para cada proyecto. Además cuenta con un sub-proyecto llamado GitHub Education que es ampliamente utilizado como herramientas para la formación de programadores (Hernández, Valladares, León, González, and González, 2018).

Sistemas de integración continua (CI)

Travis-CI

Travis CI es un servicio alojado de integración continua que se utiliza para construir y testear proyectos de software alojados en GitHub.

Los proyectos de código abierto pueden ser probados de forma gratuita a través del sitio https://travis-ci.org. En cambio los proyectos privados deben pagar para poder ser ejecutados a través del sitio https://travis-ci.com.

Gran parte de su código fuente es software libre y está disponible en GitHub.

Lenguajes utilizados

Al igual que no existe una única solución a un problema, existen multitud de lenguajes de programación, nosotros nos vamos a centrar en solamente cuatro de ellos ya que son los que cumplen con los contenidos de la mayoría de las asignaturas sobre informática, dichos lenguajes son Python, Ruby, C y HTML, aun así esta metodología es aplicable a otros lenguajes por lo que se podría utilizar para enseñar Java, Go, Haskell o Javascript:

Python

Python es un lenguaje de programación desarrollado por Guido van Rossum en 1991 cuya sintaxis favorece escribir código muy legible. Hoy en día es uno de los lenguajes de programación más utilizados en cursos de introducción a la programación ya que su sintaxis suele ser más sencilla, asemejándose al "pseudocódigo".

Según diversos índices (TIOBE Software, 2012) y encuestas (Overflow, 2019) Python es uno de los lenguajes con mayor proyección de futuro y que del que más se ha incrementado su uso, sobre todo motivado por su sencillez a la hora de aprenderlo y a su potencia como lenguaje de análisis de datos habiendo prácticamente desbancando al lenguaje estadístico R.

Vamos a centrar nuestros ejercicios en la versión 3 de Python ya que la versión 2 actualmente solo recibe actualizaciones de seguridad y en enero de 2020 dejará de estar oficialmente soportada (Python.org, 2018).

Ruby

Ruby es un lenguaje de programación orientado a objetos creado por el programador japonés Yukihiro Matsumoto en 1995. Su orientación a objetos es denominada "fuerte" ya que todos los tipos de dato son a su vez un objeto.

Ruby empezó a ganar popularidad tras la publicación del framework de aplicaciones web Ruby On Rails (RoR) por parte de David Heinemeier Hansson en 2005 ya que el mismo simplificaba muchísimo el desarrollo siguiendo el patrón Modelo Vista Controlador (MVC).

Debido a su sencilla sintaxis y su fuerte orientación a objetos es ampliamente utilizado para enseñar las particularidades de dicho paradigma.

C

El lenguaje de programación C es un lenguaje de propósito general desarrollado por Dennis Ritchie en los Laboratorios Bell entre 1969 y 1972, es el lenguaje de programación más popular para crear software para sistemas embebidos y micro-controladores, aunque también se puede utilizar para crear aplicaciones.

Es un lenguaje tipado estáticamente (en tiempo de compilación) y considerad de de medio nivel ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero permitiendo un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C, acceder a memoria o controlar diferentes dispositivos.

HTML

El lenguaje de marcas de hipertexto o HTML por sus siglas en inglés, es un lenguaje de marcado para la elaboración de páginas web. La primera versión del lenguaje fue creada por Tim Berners-Lee en 1991 mientras trabajaba en

el Centro Europeo de Investigaciones Nucleares (CERN) en Suiza. Actualmente sus especificaciones están a cargo del World Wide Web Consortium (W3C).

HTML no es un lenguaje de programación, es un lenguaje de marcado que sirve para definir documentos estandarizados.

Herramientas de análisis de código

También conocidos como "linters" son herramientas que analizan el código fuente para marcar errores de programación, *bugs*, errores estilísticos y código sospechosos. Algunos son capaces incluso de calcular la complejidad ciclomática[1] de un algoritmo. El término proviene de una antigua utilidad de Unix para análisis de código fuente en lenguaje C llamada "lint".

Existen *linters* para prácticamente todos los lenguajes de programación, en nuestros ejemplos hemos utilizado los siguientes:

Pycodestyle

Pycodestyle es una herramienta para comprobar código fuente en el lenguaje Python contra las convenciones de estilo PEP8.

PEP8 es una guía de estilo de codificación para Python definida inicialmente por Guido van Rossum, creador del lenguaje Python y que está considerada como la estándar del lenguaje.

Rubocop

Es un analizar de código para el lenguaje Ruby que tiene opciones muy interesantes como el cálculo de la complejidad ciclomática y la auto-reparación de código fuente incorrecto.

Cpplint

Herramienta de código abierto desarrollada por Google, diseñada para garantizar que el código C y C++ se ajusta a las guías de estilo de codificación de la compañía.

HTML Tidy

Es una aplicación de consola que sirve para corregir código HTML no válido, detectar posibles errores de accesibilidad web y mejorar el diseño y el estilo de sangría del marcado resultante. Fue desarrollado en 2002 por el miembro del World Wide Web Consortium (W3C) Dave Raggett.

[1] Valor escalar que mide la complejidad lógica de un programa.

Herramientas de verificación de código

Las herramientas de verificación de código se utilizan para ejecutar los test TDD o BDD en nuestro código fuente. En nuestros ejercicios vamos a utilizar las siguientes :

Pytest

Pytest es un framework de test unitarios para el lenguaje de programación Python que facilita la creación de pruebas sencillas y escalables. Las pruebas son expresivas y legibles y no se requiere código adicional.

RSpec

RSpec es una herramienta para realizar test BDD que sirve para probar código escrito en el lenguaje de programación Ruby.

MinUnit

MinUnit es un mini-framework para correr test unitarios en lenguaje C. Su código fuente se puede encontrar en http://www.jera.com/techinfo/jtns/jtn002.html. Es particularmente pequeño ya que consiste en un fichero de cabecera ".h" de tan solo 4 líneas de código.

Metodología de aprendizaje

Algunas de las herramientas necesarias para nuestro sistema de corrección requieren algo de formación previa, por lo que vamos a definir una metodología de aprendizaje para las mismas

Aprendiendo a usar Git

Como ya explicamos anteriormente Git es uno de los sistemas de control de versiones más utilizados en la actualidad. Es difícil encontrar hoy en día una empresa que se dedique a la programación de forma profesional que no la utilice. A pesar de ello no forma parte del currículo de los ciclos formativos de informática. De hecho, salvo contadas excepciones, tampoco se aprende su uso en los Grados en Ingeniería Informática. De ahí la ventaja de nuestro sistema, ya que vamos a aprovechar para enseñarles a utilizar una herramienta que van a usar de forma exhaustiva durante su futura carrera profesional.

Para aprender a utilizar Git nos vamos a basar en diversos manuales que enseñan a utilizarlo de una forma sencilla (Popov, 2012) sin necesidad de

tener conocimientos de programación. Como ya indicamos en la propuesta pedagógica vamos a definir unos contenidos comunes que son los siguientes:

Introducción a Git

En esta parte les explicaremos que es un sistema de control de versiones, que diferencias hay entre un sistema centralizado y uno distribuido. También les hablaremos de la motivación de por qué usarlos poniéndoles ejemplos de lo que seguramente ellos han utilizado hasta ahora que seguramente será duplicar los archivos en distintas carpetas. También les hablaremos un poco de la historia de Git.

Creando una cuenta en GitHub

Aquí les indicaremos como crear su primera cuenta en GitHub, para ello deberán acceder a la página web https://github.com y hacer clic en SignUp (figura 5.1).

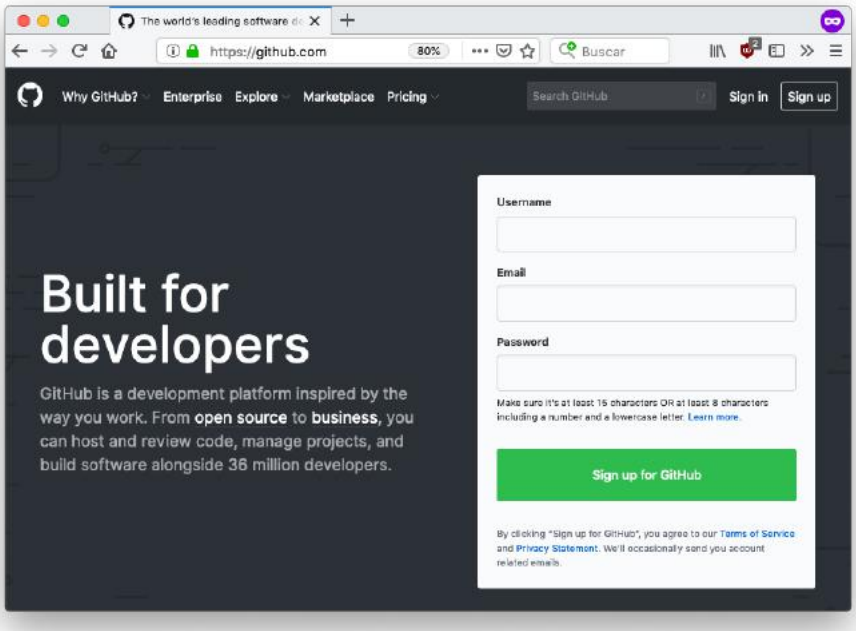

Figure 5.1: Pagina de registro de GitHub

Creando nuestro primer repositorio

Crear un repositorio en GitHub es muy intuitivo, solo tenemos que hacer clic en el botón New y rellenar los datos que nos solicita (figura 5.2). De igual

forma podemos crear un repositorio de forma local en nuestro ordenador ejecutando el comando `git init`

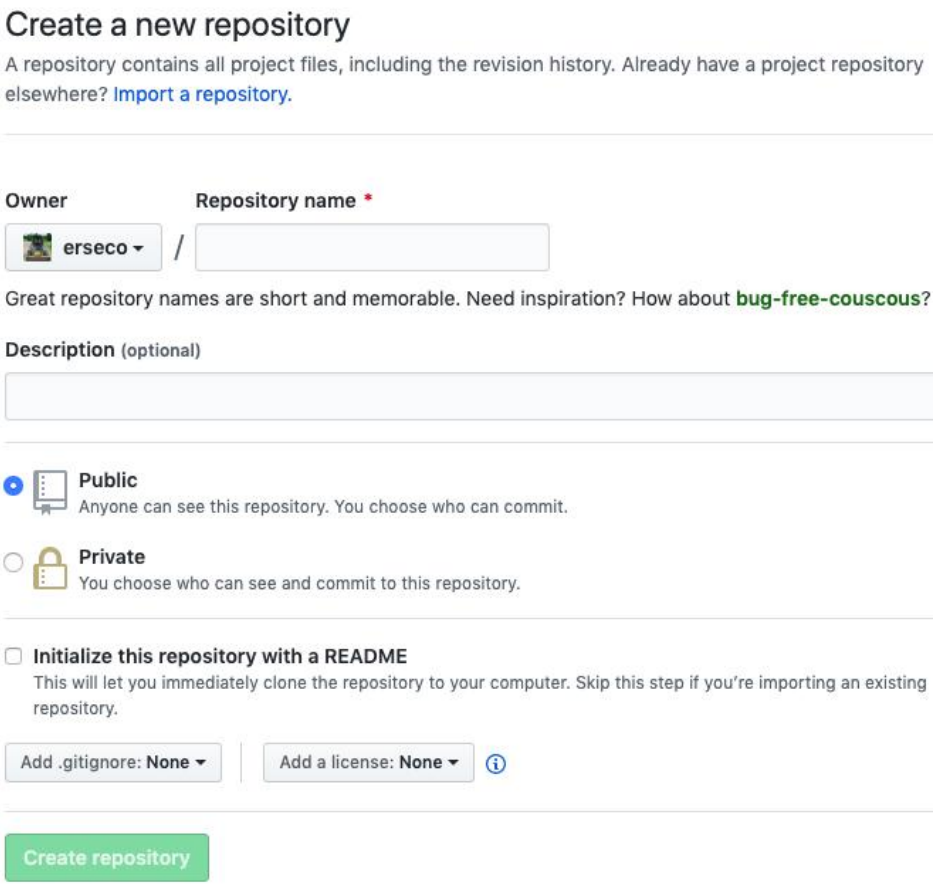

Figure 5.2: Ventana de creación de repositorio en GitHub

Para que los alumnos se familiaricen con el uso de GitHub editaremos el fichero README.md con el editor WYSIWYG[2] integrado en la plataforma.

Comandos básicos de Git

Aquí podemos ver algunos de los comandos más habituales de Git, se puede encontrar la referencia completa en su documentación oficial en https://git-scm.com/doc.

- **git clone**: Clona un repositorio
- **git status**: Nos dice el estado de un repositorio

[2] acrónimo de What You See Is What You Get (en español, "lo que ves es lo que obtienes").

- **git commit**: Nos permite guardar los cambios en una rama
- **git checkout**: Nos permite cambiar de rama
- **git branch**: Nos permite crear y listar ramas
- **git push**: Permite enviar el código a un repositorio remoto
- **git pull**: Permite obtener el código desde un repositorio remoto

Para que los alumnos se familiaricen con el uso de estos comandos vamos a pedirle que hagan cambios en los archivos de la carpeta homework$_0$0$_m$arkdownconsueditoryhag

Creando nuestro primer Fork

Una bifurcación, o fork en inglés, es el término que se utiliza para indicar una ramificación de un trabajo. Básicamente significa que vamos a copiar un proyecto y crear uno nuevo haciéndole modificaciones. La capacidad de crear bifurcaciones de código de forma sencilla es una de las características que han ayudado a la plataforma GitHub llegar a ser el sitio de referencia para albergar proyectos de software libre.

Para crear un fork en GitHub de cualquier proyecto simplemente tenemos que hacer click en el botón situado a la derecha de cada proyecto (figura 5.3). Una cosa a tener en cuenta a la hora de hacer un Fork de un proyecto es la licencia bajo la que esté dicho código, el cual suele venir indicado en el fichero LICENSE, no todas las licencias permiten la libre distribución de proyectos derivados.

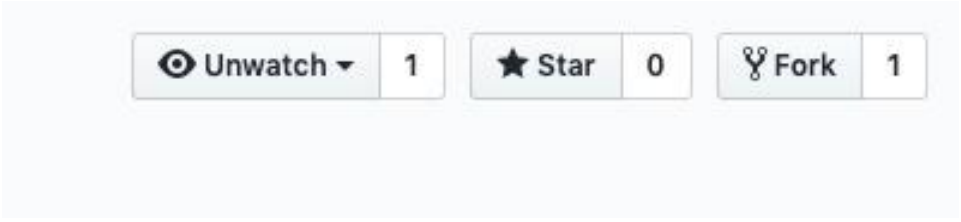

Figure 5.3: Detalle del botón de Fork en GitHub

Creando un Pull-Request

Las contribuciones a un repositorio de código fuente que utiliza un sistema de control de versiones distribuido se realizan comúnmente por medio de un “pull request”. El colaborador solicita que el encargado del proyecto haga un “pull” con los cambios en el código fuente, de ahí el nombre. El mantenedor puede revisar el conjunto de cambios, discutir modificaciones potenciales o mezclar el código.

Dependiendo del flujo de trabajo establecido el código puede ser probado antes de ser incluido en la versión oficial. Algunos proyectos ejecutan un conjunto de pruebas automatizadas en cada solicitud de extracción, utilizando una herramienta de integración continua como Travis CI, y el revisor verifica que cualquier código nuevo tenga la cobertura de pruebas adecuada.

Para hacer un "Pull request" haremos clic en el botón "New pull request" y seleccionando que ramas queremos fusionar (figura 5.4).

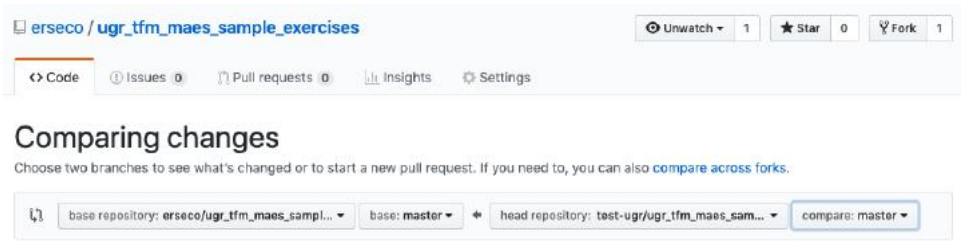

Figure 5.4: Detalle de creación de un Pull Request en GitHub

Sistemas de integración continua (CI)

Un sistema de integración continua suele consistir en una plataforma que ejecuta una serie de pasos con cada "Push" que realizamos a nuestro sistema de control de versiones. A esta serie de pasos se le suele denominar "pipeline" y suele contener pasos habituales como pueden ser la compilación, la ejecución de validaciones sintácticas y estilísticas de código (*linter*) y la ejecución de los diferentes test para comprobar que efectivamente el código funciona.

Como ya hemos indicado, para nuestra metodología vamos a utilizar "Travis CI" aunque la forma de funcionar es muy similar en casi todas las plataformas. Para crear una cuenta en "Travis CI" iremos a la url https://travis-ci.org y haremos click en el botón "Sign-Up" (figura 5.5)).

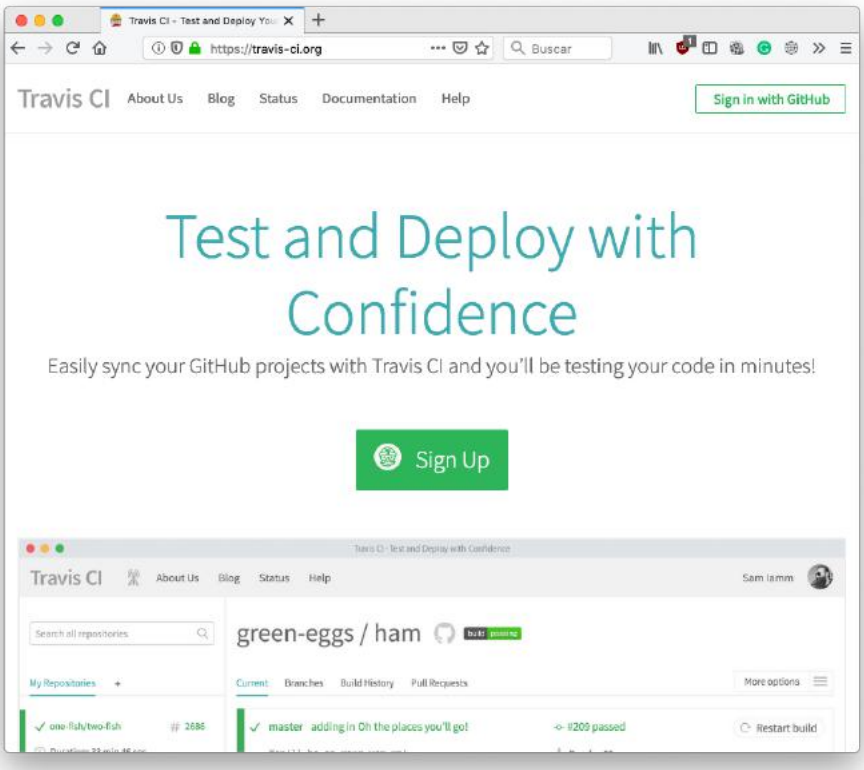

Figure 5.5: Detalle de creación de una cuenta en Travis CI

Una vez tengamos nuestra cuenta tenemos que activar en cuales de nuestra lista de repositorios queremos activar el servicio. Para ellos solo debemos hacer click en el interruptor hasta que quede de color verde (figura 5.6).

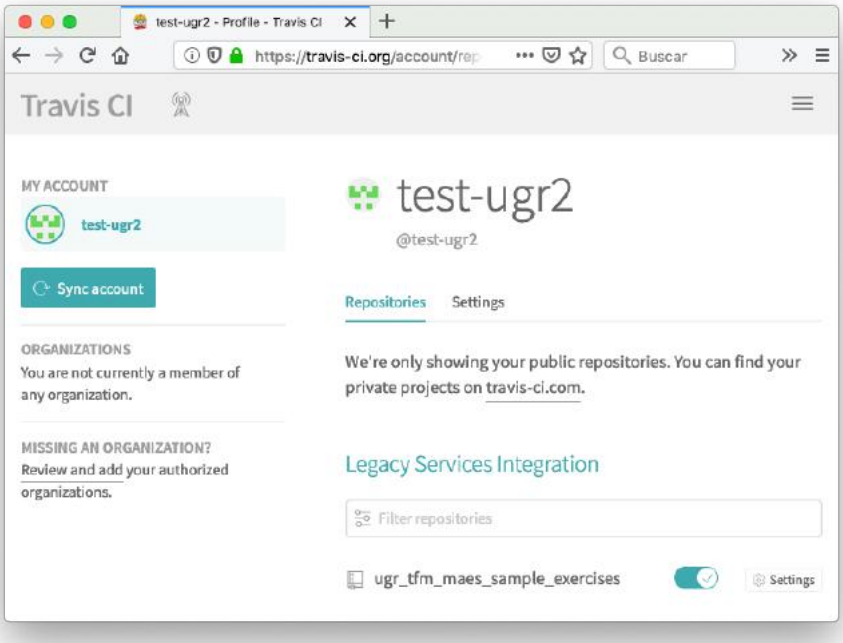

Figure 5.6: Detalle de activación de un repositorio en Travis CI

Una vez activado, con cada `git push` se ejecutará el "pipeline" definido en el fichero `.travis.yml`, en nuestro repositorio de ejemplo se puede obtener uno configurado para ejecutar los diferentes ejemplos realizados para mostrar el funcionamiento de la metodología.

Introducción a la programación

Una vez los alumnos han aprendido los comandos básicos de Git y se han dado de alta en GitHub es hora de enseñarles algunos conceptos básicos de programación.

Conceptos básicos de programación

Todos los lenguajes de programación comparten algunos elementos básicos que funcionan y se usan de forma diferente en cada lenguaje, pero que cumplen el mismo objetivo. Esos elementos son:

- Tipos de datos: Enteros, Decimales, Caracteres, Cadenas de texto, etc...
- Variables: donde almacenar los datos.

- Control de flujo: los “if”
- Bucles: los conocidos “loops”, “for” y “while”
- Funciones
- Entra y Salida: pintando en pantalla.

Hay que remarcar que estamos explicando los conceptos básicos de programación, por eso esto no incluye ni estructuras de datos, ni orientación a objetos ni recursividad, realmente eso lo aprenderán en ejercicios adicionales.

Nuestro primer “Hola Mundo”

Un “Hola mundo” es un programa cuya única finalidad es escribir la frase “Hola mundo!”. Este programa se usa como introducción en la mayoría de lenguajes de programación siendo el primer ejercicio típico, y se considera fundamental desde el punto de vista didáctico. Una implementación de dicho programa se puede encontrar para prácticamente todos los lenguajes de programación existentes.

En nuestra metodología animamos a los profesores a enseñar un primer ejemplo del lenguaje a impartir usando un “Hola Mundo” para que los alumnos se familiaricen con el lenguaje. En nuestros ejercicios de ejemplo vamos a incorporar el hola mundo como primer ejercicio de programación.

Guías de estilo

En esta sección les enseñaremos la guía de estilo que vamos a seguir. Aunque es algo que se suele obviar en cursos de programación, a la hora de escribir código de calidad es importante seguir una guía de estilo, además de crear buenas prácticas de programación. Les explicaremos lo importante que es comentar correctamente y se utilizará a poder ser una guía de estilo estandarizada para el lenguaje que vayamos a impartir.

Si el lenguaje que vamos a impartir no tuviera guía de estilo estandarizada, o si el profesor lo estima didáctico se podría consensuar con los alumnos el tipo de estilo que se va a seguir. Este caso es aplicable al uso de comillas simples ’ o dobles ”” pues no hay un consenso sobre cuales se deben usar, igual pasa con la indentación con espacios o con tabuladores.

Introducción al TDD

Aquí les explicaremos en consiste el TDD, que como ya hemos visto es escribir la prueba, escribir el código y una vez funcione refactorizar. En nuestro caso concreto les vamos a dar nosotros realizadas las pruebas TDD, pero les podremos animar a que realicen pruebas adicionales.

Ejercicios de ejemplo

En el repositorio https://github.com/erseco/ugr_tfm_maes_sample_exercises hemos definido una serie de ejercicios así como sus correcciones, dichos ejercicios se han incorporado en la sección de Anexos.

Usando la integración continua para aprender

Una vez hemos visto como usar Git y los conceptos básicos de programación vamos a ver algunos de los errores con los que se pueden encontrar los alumnos, como pueden usar el sistema de integración continua para detectarlos ellos mismos y como corregirlos (figura 5.7).

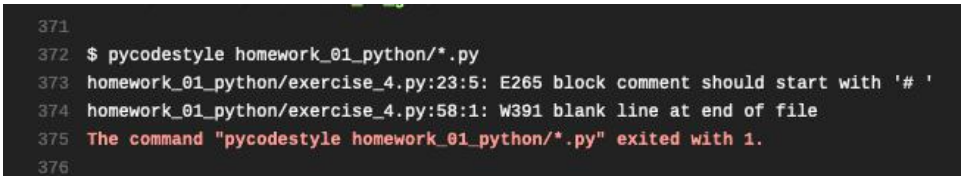

Figure 5.7: Detalle de un error de linter python en un pipeline

Chapter 6

Conclusiones

Tras el análisis de las numerosas herramientas de corrección y verificación existentes, así como la retroalimentación recibida parte de profesores y alumnos, sin olvidar a las personas que respondieron a la encuesta realizada he concluido que, a pesar de tener algunos inconvenientes, los sistema de autocorrección nos proveen de una serie de ventajas muy a tener en cuenta.

Aun así, dichos sistemas tienen algunos inconvenientes. *Coderunner*, por ejemplo, depende de que tengamos una instalación de Moodle, y *Pythontutor* está limitado al aprendizaje del lenguaje Python.

Además, dichos sistemas existentes, necesitan de un aprendizaje previo.

Nuestro sistema tiene la ventaja de utilizar tecnologías estándares con amplia difusión en la industria informática con lo que no van a aprender a usar una herramienta que no usarán jamás cuando finalicen la asignatura, todo lo contrario, les vamos a enseñar a usar tecnologías con las que se van a tener que enfrentar más tarde o más temprano.

En mis años de Universidad jamás me hablaron de los *linters*, de las guías de estilo de codificación y de las pruebas unitarias (TDD) se vio de pasada. Tampoco aparece el uso de estas herramientas en ninguna de las asignaturas relacionadas con la informática de Educación Secundaria, Bachillerato y Formación Profesional. Hay una asignatura llamada "Entornos de desarrollo" que enseña a usar algunos IDEs[1] no contempla ninguna de las tecnologías mencionadas.

Esta metodología sirve para enseñar cualquier lenguaje de programación o paradigma de programación existente, es decir, es completamente agnóstica del lenguaje.

Al guardarse un registro de todo lo que se envía al repositorio de código junto al resultado de su compilación, el profesor puede tener métricas muy detalladas de los errores cometidos por cada alumno, ver su evolución e incluso el tiempo dedicado.

Observando el *pipeline* los profesores pueden ver de forma global donde

[1] Entorno de Desarrollo Integrado, Integrated Development Environment en inglés.

están fallando los alumnos y realizar modificaciones sobre los ejercicios para incidir en los temas en los que los alumnos tengan mayor dificultad.

Además, al ser software libre y estar disponible de forma pública cualquiera puede hacer uso de los ejercicios e incluso proponer mejoras, a su vez los alumnos pueden enseñar su código en futuras entrevistas de trabajo. A día de hoy el mejor currículum vitae de un programador informático es su perfil en GitHub.

Como inconvenientes podemos destacar que gran parte de los docentes tampoco están habituados a usar los sistemas utilizados por lo que deberían formarse en su utilización. Además la curva de trabajo para el profesor es más pronunciada pues tiene que preparar todos los ejercicios gran variedad de ejercicios y test antes del comienzo de las clases. Aunque, eso sí, descarga al profesor de tener que corregir.

Otro inconveniente es que el profesor debe ir modificando regularmente los ejercicios ya que al estar disponibles de forma pública una simple búsqueda les permitiría a los alumnos obtener la resolución de los ejercicios de cursos anteriores.

Aun con estos inconvenientes vemos que las ventajas son mucho mayores, y sumándole la retroalimentación recibida parte de profesores y alumnos además de las personas que respondieron a la encuesta podemos concluir que nuestro sistema aporta un valor añadido evidente a los sistemas existentes de corrección y espero implantar la metodología en mis propias clases.

Chapter 7

Anexos

Código fuente

Pipeline para Travis-CI

```
---
dist: xenial

before_install:
  - pyenv global system 3.6.7
  - sudo apt-get update
  - sudo apt-get install -y tidy cppcheck
install:
  - pip3 install pytest pycodestyle requests beautifulsoup4
  - gem install rspec rubocop mdl

script:
  - echo "Testing that we have all the requirements"
  - mdl --version
  - python3 --version
  - ruby --version
  - gcc --version
  - tidy --version

  - mdl homework_00_git/

  - pycodestyle homework_01_python/*.py
  - pytest --verbose homework_01_python/*.py

  - rubocop homework_02_ruby
  - rspec homework_02_ruby

  - cppcheck --verbose --error-exitcode=2 homework_03_c/*.c
  - make -C homework_03_c tests

  - tidy -quiet -errors homework_04_html/*.html
```

Fragmento de código 7.1: .travis.yml

Ejercicios Git/Markdown

```
# Hello world
```

Fragmento de código 7.2: exercise_01.md

```
#h1 Heading

## h2 Heading

### h3 Heading

#### h4 Heading

##### h5 Heading

## Horizontal Rule
---


## Emphasis

Text **This is bold text**

Text *This is italic text*

Text _This is italic text_

Text ~~Strikethrough~~

## Lists

- Create a list by starting a ‘-‘
- Very easy!

## Code

‘‘‘
Sample text here...
‘‘‘

## Tables

| name   | description |
| ------ | ----------- |
| name1  | description1. |
| name2  | description2. |
| name3  | description3. |

## Links

[link text](http://www.google.es)

[link with title](http://www.google.es/ "title text!")
```

Fragmento de código 7.3: exercise_02.md

Ejercicios Python

```python
#!/usr/bin/env python3

# Question:
# Write a program that prints "Hello world!"
#
# Hints:
# Use the python print() built-infunction


print('Hola mundo!')
```

Fragmento de código 7.4: exercise_1.py

```python
#!/usr/bin/env python3

# Question:
# Write a function called add() which can compute and returns the summ
    of two
# given numbers.
#
# Suppose the following parameters are supplied to the program:
# 8 8
# Then, the output should be:
# 16

import sys


def add(x, y):
    return x + y


# ------- START TDD TESTS DEFINITION -----------
def test_add_8_8():
    assert add(8, 8) == 16


def test_add_4_5():
    assert add(4, 5) == 9
# ------- END TDD TESTS DEFINITION -------------


# Program entrypoint
if __name__ == "__main__":

    if len(sys.argv) == 3:

        number1 = int(sys.argv[1])
        number2 = int(sys.argv[2])
        result = add(number1, number2)
        print(result)

    else:
        print("Usage: %s <number1> <number2>" % sys.argv[0])
```

Fragmento de código 7.5: exercise_2.py

```python
#!/usr/bin/env python3

# Question:
# Write a function called get_factorial() which can compute and returns the
# factorial of a given number.
#
# Suppose the following parameter is supplied to the program:
# 8
# Then, the output should be:
# 40320
#
# Hints:
# You can't use the `math` module.

import sys


def get_factorial(number):
    if number == 0:
        return 1
    return number * get_factorial(number - 1)


# ------- START TDD TESTS DEFINITION -----------
def test_get_factorial_8():
    import math
    assert math.factorial(8) == get_factorial(8)


def test_get_factorial_3():
    import math
    assert math.factorial(3) == get_factorial(3)


def test_get_factorial_25():
    import math
    assert math.factorial(25) == get_factorial(25)
# ------- END TDD TESTS DEFINITION -------------


# Program entrypoint
if __name__ == "__main__":

    if len(sys.argv) == 2:
        number = int(sys.argv[1])
        result = get_factorial(number)
        print(result)

    else:
        print("Usage: %s <number>" % sys.argv[0])
```

Fragmento de código 7.6: exercise_3.py

```python
#!/usr/bin/env python3

# Question:
# Write a function that get the content of the <title> tag of a given url
```

```
#
# Suppose the following parameter is supplied to the program:
# https://www.google.es
# Then, the output should be:
# Google
#
# Hints:
# Use the requests module to read the web content
# Use the BeautifulSoup to process the HTML

import sys
import requests
from bs4 import BeautifulSoup


def get_title(url):

    r = requests.get(url)
    soup = BeautifulSoup(r.text, features="html.parser")

    return soup.title.contents[0]


# ------- START TDD TESTS DEFINITION -----------
def test_get_title_google():
    url = "https://www.google.com/"
    assert get_title(url) == "Google"


def test_get_title_alhambra():
    url = "http://www.alhambra-patronato.es/"
    assert get_title(url) == "Patronato de la Alhambra y Generalife"


def test_get_title_wikipedia():
    url = "https://en.wikipedia.org/wiki/Main_Page"
    assert get_title(url) == "Wikipedia, the free encyclopedia"
# ------- END TDD TESTS DEFINITION -------------


# Program entrypoint
if __name__ == "__main__":

    if len(sys.argv) == 2:
        url = sys.argv[1]
        result = get_title(url)
        print(result)

    else:
        print("Usage: %s <url>" % sys.argv[0])
```

Fragmento de código 7.7: exercise_4.py

```
#!/usr/bin/env python3

# Question:
# Write a funciton that works like a "Rock-Paper-Scissors" game,
    remember the
# rules:
# - Rock beats scissors
```

```
7  # - Scissors beats paper
8  # - Paper beats rock
9  #
10 # The result should be:
11 # - It's a tie!
12 # - Rock wins!
13 # - Scissors win!
14 # - Paper wins!
15 # - Invalid input!
16 #
17 # Suppose the following parameter is supplied to the program:
18 # rock scissors
19 # Then, the output should be:
20 # Rock wins!
21
22
23 def play_game(option1, option2):
24
25     if option1 == option2:
26         return("It's a tie!")
27     elif option1 == 'rock':
28         if option2 == 'scissors':
29             return("Rock wins!")
30         else:
31             return("Paper wins!")
32     elif option1 == 'scissors':
33         if option2 == 'paper':
34             return("Scissors win!")
35         else:
36             return("Rock wins!")
37     elif option1 == 'paper':
38         if option2 == 'rock':
39             return("Paper wins!")
40         else:
41             return("Scissors win!")
42     else:
43         return("Invalid input!")
44
45
46 # ------- START TDD TESTS DEFINITION -----------
47 def test_play_game_rock_scissors():
48     assert play_game("rock", "scissors") == "Rock wins!"
49
50
51 def test_play_game_scissors_scissors():
52     assert play_game("scissors", "scissors") == "It's a tie!"
53
54
55 def test_play_game_paper_rock():
56     assert play_game("paper", "rock") == "Paper wins!"
57
58
59 def test_play_game_scissors_paper():
60     assert play_game("scissors", "paper") == "Scissors win!"
61
62
63 def test_play_game_invalid_paper():
64     assert play_game("invalid", "paper") == "Invalid input!"
65 # ------- END TDD TESTS DEFINITION -------------
66
67
68 # Program entrypoint
```

```
69 if __name__ == "__main__":
70
71     if len(sys.argv) == 3:
72
73         option1 = sys.argv[1]
74         option2 = sys.argv[2]
75         result = play_game(option1, option2)
76         print(result)
77
78     else:
79         print("Usage: %s <option1> <option2>" % sys.argv[0])
```

Fragmento de código 7.8: exercise_5.py

Ejercicios Ruby

```
1  #!/usr/bin/env ruby
2  # frozen_string_literal: true
3
4  # Question:
5  # Write a program that prints "Hello world!"
6  #
7  # Hints:
8  # Use the ruby puts directive
9
10 puts 'Hello world!'
```

Fragmento de código 7.9: exercise_1.rb

```
1  #!/usr/bin/env ruby
2  # frozen_string_literal: true
3
4  # Question:
5  # Write a function called add() which can compute and returns the summ
       of two
6  # given numbers.
7  #
8  # Suppose the following parameters are supplied to the program:
9  # 8 8
10 # Then, the output should be:
11 # 16
12
13 def add(number1, number2)
14   number1 + number2
15 end
16
17 # Program entrypoint
18 if $PROGRAM_NAME == __FILE__
19   if ARGV.length == 2
20
21     number1 = ARGV[0].to_i
22     number2 = ARGV[1].to_i
23     result = add(number1, number2)
24     puts result
25   else
26     puts "Usage: #{$PROGRAM_NAME} <number1> <number2>"
27   end
```

```
28 end
```

Fragmento de código 7.10: exercise_2.rb

```
1  # frozen_string_literal: true
2
3  require_relative '../exercise_2'
4
5  # ------- START TDD TESTS DEFINITION -----------
6  describe add(0, 0) do
7    it 'returns 16 when passed 8 8' do
8      expect(add(8, 8)).to eq 16
9    end
10
11   it 'returns 9 when passed 4 5' do
12     expect(add(4, 5)).to eq 9
13   end
14 end
15 # ------- END TDD TESTS DEFINITION -------------
```

Fragmento de código 7.11: exercise_2_spec.rb

```
1  #!/usr/bin/env ruby
2  # frozen_string_literal: true
3
4  # Question
5  # Write a function called get_factorial() which can compute and
       returns the
6  # factorial of a given number.
7  #
8  # Suppose the following parameter is supplied to the program
9  # 8
10 # Then, the output should be
11 # 40320
12 #
13 # Hints:
14 # You can't use the `math` module.
15
16 def get_factorial(number)
17   return 1 if number.zero?
18
19   number * get_factorial(number - 1)
20 end
21
22 # Program entrypoint
23 if $PROGRAM_NAME == __FILE__
24
25   if ARGV.length == 1
26     number = ARGV[0].to_i
27     result = get_factorial(number)
28     puts result
29   else
30     puts "Usage: #{$PROGRAM_NAME} <number>"
31   end
32 end
```

Fragmento de código 7.12: exercise_3.rb

```ruby
# frozen_string_literal: true

require_relative '../exercise_3'

# ------- START TDD TESTS DEFINITION -----------
describe get_factorial(0) do
  it 'returns 40320 when passed 8' do
    expect(get_factorial(8)).to eq 40_320
  end

  it 'returns 6 when passed 3' do
    expect(get_factorial(3)).to eq 6
  end
  it 'returns 15511210043330985984000000 when passed 3' do
    expect(get_factorial(25)).to eq 15_511_210_043_330_985_984_000_000
  end
end
# ------- END TDD TESTS DEFINITION -------------
```

Fragmento de código 7.13: exercise_3_spec.rb

```ruby
#!/usr/bin/env ruby

# Question:
# Write a function that get the content of the <title> tag of a given
    url
#
# Suppose the following parameter is supplied to the program:
# https://www.google.es
# Then, the output should be:
# Google
#
# Hints:
# Use the open-uri module to read the web content
# Use a regular expression to process the HTML
require "open-uri"

def get_title(url)
    open(url) do |f|
        str = f.read
        str.scan(/<title>(.*?)<\/title>/)
    end
end

# Program entrypoint
if __FILE__ == $PROGRAM_NAME

    if ARGV.length == 1
        url = ARGV[0]
        result = get_title(url)
        puts result

    else
        puts "Usage: #{$PROGRAM_NAME} <url>"
    end
end
```

Fragmento de código 7.14: exercise_4.rb

```ruby
# frozen_string_literal: true

require_relative '../exercise_4'

# ------- START TDD TESTS DEFINITION -----------
#  RSpec  don't allow to connect to external websites, so we should mock the petition
describe get_factorial(0) do
    it 'returns "Google" when passed "https//www.google.com/"' do
      expect(get_title('https//www.google.com/')).to eq 'Google'
    end
    it 'returns "Patronato de la Alhambra y Generalife" when passed "http//www.alhambra-patronato.es/"' do
      expect(get_title('http//www.alhambra-patronato.es/')).to eq 'Patronato de la Alhambra y Generalife'
    end
    it 'returns "Wikipedia, the free encyclopedia" when passed "https//en.wikipedia.org/wiki/Main_Page"' do
      expect(get_title('https//en.wikipedia.org/wiki/Main_Page')).to eq 'Wikipedia, the free encyclopedia'
    end
end
# ------- END TDD TESTS DEFINITION -------------
```

Fragmento de código 7.15: exercise_4_spec.rb

```ruby
#!/usr/bin/env ruby
# frozen_string_literal: true

# Question:
# Write a funciton that works like a "Rock-Paper-Scissors" game, remember the
# rules:
# - Rock beats scissors
# - Scissors beats paper
# - Paper beats rock
#
# The result should be:
# - It's a tie!
# - Rock wins!
# - Scissors win!
# - Paper wins!
# - Invalid input!
#
# Suppose the following parameter is supplied to the program:
# rock scissors
# Then, the output should be:
# Rock wins!

def play_game(option1, option2)
  if option1 == option2
    "It's a tie!"
  elsif option1 == 'rock'
    if option2 == 'scissors'
      'Rock wins!'
    else
      'Paper wins!'
    end
  elsif option1 == 'scissors'
```

```
33      if option2 == 'paper'
34        'Scissors win!'
35      else
36        'Rock wins!'
37      end
38    elsif option1 == 'paper'
39      if option2 == 'rock'
40        'Paper wins!'
41      else
42        'Scissors win!'
43      end
44    else
45      'Invalid input!'
46    end
47  end
48
49  # Program entrypoint
50  if $PROGRAM_NAME == __FILE__
51
52    if ARGV.length == 2
53
54      option1 = ARGV[0]
55      option2 = ARGV[1]
56      result = play_game(option1, option2)
57      puts result
58
59    else
60      puts "Usage: #{$PROGRAM_NAME} <option1> <option2>"
61    end
62  end
```

Fragmento de código 7.16: exercise_5.rb

```
1   # frozen_string_literal: true
2
3   require_relative '../exercise_5'
4
5   # ------- START TDD TESTS DEFINITION -----------
6   describe play_game('', '') do
7     it 'returns "Rock wins!" when passed "rock" "scissors"' do
8       expect(play_game('rock', 'scissors')).to eq 'Rock wins!'
9     end
10
11    it 'returns "Its a tie!" when passed "scissors" "scissors"' do
12      expect(play_game('scissors', 'scissors')).to eq "It's a tie!"
13    end
14
15    it 'returns "Paper wins!" when passed "paper" "rock"' do
16      expect(play_game('paper', 'rock')).to eq 'Paper wins!'
17    end
18
19    it 'returns "Scissors win!" when passed ("scissors" "paper"' do
20      expect(play_game('scissors', 'paper')).to eq 'Scissors win!'
21    end
22
23    it 'returns "Invalid input!" when passed "invalid" "paper"' do
24      expect(play_game('invalid', 'paper')).to eq 'Invalid input!'
25    end
26  end
27  # ------- END TDD TESTS DEFINITION -------------
```

Fragmento de código 7.17: exercise_5_spec.rb

Ejercicios C

```
SRCS = $(wildcard *.c)
PROGS = $(patsubst %.c,%,$(SRCS))

all: $(PROGS)

%: %.c
  gcc $(CFLAGS) -o $@ $<

tests: test_2 test_3 test_5 clean

test_2:
  @gcc $(CFLAGS) -Wl, -e_main_tests -o exercise_2 exercise_2.c
  @./exercise_2

test_3:
  @gcc $(CFLAGS) -Wl, -e_main_tests -o exercise_3 exercise_3.c
  @./exercise_3

test_5:
  @gcc $(CFLAGS) -Wl, -e_main_tests -o exercise_5 exercise_5.c
  @./exercise_5

clean:
  @rm -f $(PROGS)
```

Fragmento de código 7.18: Makefile

```
 /* file: minunit.h */
 #define mu_assert(message, test) do { if (!(test)) return message; }
     while (0)
 #define mu_run_test(test) do { char *message = test(); tests_run++; \
                                if (message) return message; } while
                                    (0)
int tests_run;

char * all_tests();

// This function acts as entrypoint for TDD
int main_tests(int argc, char **argv) {
     char *result = all_tests();
     if (result != 0) {
         printf("%s\n", result);
     }
     else {
         printf("ALL TESTS PASSED on %s\n", argv[0]);
     }
     printf("Tests run: %d\n", tests_run);

     return result != 0;
}
```

Fragmento de código 7.19: minunit.h

```c
/*
# Question:
# Write a program that prints "Hello world!"
#
# Hints:
# Use the c printf() built-infunction
*/

#include <stdio.h>

int main(void)
{
    printf("Hello world!\n");
    return 0;
}
```

Fragmento de código 7.20: exercise_1.c

```c
/*
# Question:
# Write a function called add() which can compute and returns the summ
    of two
# given numbers.
#
# Suppose the following parameters are supplied to the program:
# 8 8
# Then, the output should be:
# 16
*/

#include <stdio.h> //printf, scanf
#include <stdlib.h> //atoi, strtol
#include "minunit.h"

int add(int x, int y) {
    return x + y;
}

// ------- START TDD TESTS DEFINITION -----------
char * test_add_8_8(){
    mu_assert("error", add(8, 8) == 16);
    return 0;
}

char * test_add_4_5() {
    mu_assert("error", add(4, 5) == 9);
    return 0;
}

char * all_tests() {
     mu_run_test(test_add_8_8);
     mu_run_test(test_add_4_5);
     return 0;
 }
// ------- END TDD TESTS DEFINITION -------------
```

```
37
38 // Program entrypoint
39 int main(int argc, char **argv)
40 {
41
42     if (argc == 3) {
43
44         int number1 = atoi(argv[1]);
45         int number2 = atoi(argv[2]);
46         int result = add(number1, number2);
47         printf("%d\n", result);
48
49     } else {
50         printf("Usage: %s <number1> <number2>\n", argv[0]);
51     }
52
53     return 0;
54 }
```

Fragmento de código 7.21: exercise_2.c

```
1  /*
2  # Question:
3  # Write a function called get_factorial() which can compute and
       returns the
4  # factorial of a given number.
5  #
6  # Suppose the following parameter is supplied to the program:
7  # 8
8  # Then, the output should be:
9  # 40320
10 #
11 # Hints:
12 # You can't use the `math` module.
13 */
14
15 #include <stdio.h> //printf, scanf
16 #include <stdlib.h> //atoi, strtol
17 #include "minunit.h"
18
19
20 int get_factorial(int number) {
21     if (number == 0) {
22         return 1;
23     }
24     return number * get_factorial(number - 1);
25 }
26
27 // ------- START TDD TESTS DEFINITION -----------
28 char * test_get_factorial_8(){
29     mu_assert("error", get_factorial(8) == 40320);
30     return 0;
31 }
32
33 char * test_get_factorial_3() {
34     mu_assert("error", get_factorial(3) == 6);
35     return 0;
36 }
37
38 char * test_get_factorial_10() {
```

```
39      mu_assert("error", get_factorial(10) == 3628800);
40      return 0;
41  }
42
43  char * all_tests() {
44       mu_run_test(test_get_factorial_8);
45       mu_run_test(test_get_factorial_3);
46       mu_run_test(test_get_factorial_10);
47       return 0;
48   }
49  // ------- END TDD TESTS DEFINITION -------------
50
51  // Program entrypoint
52  int main(int argc, char **argv)
53  {
54
55      if (argc == 2) {
56
57          int number = atoi(argv[1]);
58          int result = get_factorial(number);
59          printf("%d\n", result);
60
61      } else {
62          printf("Usage: %s <number>\n", argv[0]);
63      }
64
65      return 0;
66  }
```

Fragmento de código 7.22: exercise_3.c

```
1   /*
2   # Question:
3   # Write a funciton that works like a "Rock-Paper-Scissors" game,
        remember the
4   # rules:
5   # - Rock beats scissors
6   # - Scissors beats paper
7   # - Paper beats rock
8   #
9   # The result should be:
10  # - It's a tie!
11  # - Rock wins!
12  # - Scissors win!
13  # - Paper wins!
14  # - Invalid input!
15  #
16  # Suppose the following parameter is supplied to the program:
17  # rock scissors
18  # Then, the output should be:
19  # Rock wins!
20  */
21
22  #include <stdio.h> //printf, scanf
23  #include <string.h>
24  #include "minunit.h"
25
26  int add(int x, int y) {
27      return x + y;
28  }
```

```
29
30 char * play_game(char * option1, char * option2) {
31
32     if (strcmp(option1, option2))
33     {
34         return("It's a tie!");
35     }
36     else if (strcmp(option1, "rock"))
37     {
38         if (strcmp(option2, "scissors"))
39         {
40             return("Rock wins!");
41         }
42         else
43         {
44             return("Paper wins!");
45         }
46     }
47     else if (strcmp(option1, "scissors"))
48     {
49         if (strcmp(option2, "paper"))
50         {
51             return("Scissors win!");
52         }
53         else
54         {
55             return("Rock wins!");
56         }
57     }
58     else if (strcmp(option1, "paper"))
59     {
60         if (strcmp(option2, "rock"))
61         {
62             return("Paper wins!");
63         }
64         else
65         {
66             return("Scissors win!");
67         }
68     }
69     else
70     {
71         return("Invalid input!");
72     }
73
74 }
75
76 // ------- START TDD TESTS DEFINITION -----------
77
78 char * test_play_game_rock_scissors(){
79     mu_assert("error", strcmp(play_game("rock", "scissors"), "Rock
          wins!"));
80     return 0;
81 }
82
83
84 char * test_play_game_scissors_scissors(){
85     mu_assert("error", strcmp(play_game("scissors", "scissors"), "It's
           a tie!"));
86     return 0;
87 }
88
```

```
89
90  char * test_play_game_paper_rock(){
91      mu_assert("error", strcmp(play_game("paper", "rock"), "Paper wins!
            "));
92      return 0;
93  }
94
95
96  char * test_play_game_scissors_paper(){
97      mu_assert("error", strcmp(play_game("scissors", "paper"), "
            Scissors win!"));
98      return 0;
99  }
100
101
102 char * test_play_game_invalid_paper(){
103     mu_assert("error", strcmp(play_game("invalid", "paper"), "Invalid
            input!"));
104     return 0;
105 }
106
107 char * all_tests() {
108      mu_run_test(test_play_game_rock_scissors);
109      mu_run_test(test_play_game_scissors_scissors);
110      mu_run_test(test_play_game_paper_rock);
111      mu_run_test(test_play_game_scissors_paper);
112      mu_run_test(test_play_game_invalid_paper);
113
114      return 0;
115  }
116 // ------- END TDD TESTS DEFINITION -------------
117
118 // Program entrypoint
119 int main(int argc, char **argv)
120 {
121
122     if (argc == 3) {
123
124         char * option1 = argv[1];
125         char * option2 = argv[2];
126         char * result = play_game(option1, option2);
127         printf("%s\n", result);
128
129     } else {
130         printf("Usage: %s <option1> <option2>\n", argv[0]);
131     }
132
133     return 0;
134 }
```

Fragmento de código 7.23: exercise_5.c

Ejercicios HTML

```
1  <!--
2  Question:
3  Write a program that prints "Hello world!"
4
5  Hints:
```

```
Put your text directly inside the <body> tag
-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
    TR/html4/strict.dtd">
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  Hello world!
</body>
</html>
```

Fragmento de código 7.24: exercise_1.html

```
<!--
Question:

Create a simple page with a heading section and a paragraph
-->
<!DOCTYPE html>
<head>
    <title>exercise2</title>
</head>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

Fragmento de código 7.25: exercise_2.html

```
<!--
Question:
Create a html 4x4 html table and fill with data
-->
<!DOCTYPE html>
<html>
<head>
    <title>March Bills</title>
</head>

<body>

<h2>March Bills</h2>

<table summary="March bills">
    <tr>
        <th></th>
        <th>Price</th>
        <th>Due Date</th>
    </tr>

    <tr>
        <td>Phone</td>
```

```
24          <td>$50</td>
25          <td>March 1st</td>
26      </tr>
27
28      <tr>
29          <td>Car insurance</td>
30          <td>$100</td>
31          <td>March 5th</td>
32      </tr>
33
34      <tr>
35          <td>Internet</td>
36          <td>$70</td>
37          <td>March 10th</td>
38      </tr>
39  </table>
40
41  </body>
42  </html>
```

Fragmento de código 7.26: exercise_3.html

Bibliography

Barr, V. & Trytten, D. (yearmonthday). Using turing's craft CodeLab to support CS1 students as they learn to program. *ACM Inroads*. **retrieved from** https : / / www . researchgate . net / publication / 303319606_Using_Turing ' s_Craft_CodeLab_to_support_CS1_students_as_they_learn_to_program

Benotti, L., Aloi, F., Bulgarelli, F., & Gomez, M. J. (yearmonthday). The effect of a web-based coding tool with automatic feedback on students' performance and perceptions. In *Proceedings of the 49th ACM technical symposium on computer science education* (pp. 2–7). SIGCSE '18. event-place: Baltimore, Maryland, USA. New York, NY, USA: ACM. doi:10.1145/3159450.3159579

Bruce, C. S. (yearmonthday). Contemporary developments in teaching and learning introductory programming: towards a research proposal. *Teaching and Learning Report*. **retrieved from** https : / / www . academia . edu / 18530488 / Contemporary_developments_in_teaching_and_learning_introductory_programming_Towards_a_research_proposal

Comons, C. (yearmonthday). Creative commons - reconocimiento 4.0 internacional - CC BY 4.0. **retrieved from** https : / / creativecommons.org/licenses/by/4.0/legalcode.es

de Andalucía, J. (yearmonthday). Decreto 111/2016, de 14 de junio, por el que se establece la ordenación y el currículo de la educación secundaria obligatoria en la comunidad autónoma de andalucía. BOJA. **retrieved from** https://www.juntadeandalucia.es/boja/2016/122/2

FECYT & Everis. (yearmonthday). La educación en Ciencias de la Computación en España 2015, a debate [FECYT]. **retrieved from** https://www.fecyt.es/es/publicacion/educacion-de-las-ciencias-de-la-computacion-en-espana

Godse, S. (yearmonthday). *Fail fast, succeed faster: lessons on how to avoid business failure : inspired by real life stories of failures and challenges*. OCLC: 987318427. **retrieved** urlyearurlmonthurlday, **from** http://proxy.uqtr.ca/login.cgi?action=login&u=uqtr&db=

ebsco&ezurl=http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=658154

Guo, P. J. (yearmonthday). Online Python Tutor: embeddable web-based program visualization for CS education. In *Proceedings of the 44th acm technical symposium on computer science education* (pp. 579–584). SIGCSE '13. Denver, Colorado, USA: ACM. doi:10.1145/2445196.2445368

Hernández, C. L., Valladares, G. M., León, C. R., González, E. S., & González, C. S. (yearmonthday). Integración de las herramientas "github education" en el aula. In *De la innovación imaginada a los procesos de cambio, 2018, ISBN 978-84-15939-62-7, págs. 333-346* (pp. 333–346). De la innovación imaginada a los procesos de cambio. Servicio de Publicaciones.

Ismail, S. (yearmonthday). *Exponential organizations: why new organizations are ten times better, faster, and cheaper than yours (and what to do about it)*. Diversion Books.

Ismail, S., Palao, F., & Lapierre, M. (yearmonthday). *Exponential transformation: the ExO sprint playbook to evolve your organization to navigate industry disruption and change the world for the better*. Diversion Books.

Martin, R. C. (Ed.). (yearmonthday). *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ: Prentice Hall.

GNU General Public License. (yearmonthday). Version 3. Free Software Foundation. **retrieved from** http://www.gnu.org/licenses/gpl.html

How GitHub Classroom and Travis CI improved students' grades. (yearmonthday). **retrieved from** https://github.blog/2019-02-12-how-github-classroom-and-travis-ci-improved-students-grades/

Overflow, S. (yearmonthday). Stack overflow developer survey 2019 [Stack overflow]. **retrieved** urlyearurlmonthurlday, **from** https://insights.stackoverflow.com/survey/2019/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2019

Popov, D. (yearmonthday). Control de versiones: git para no programadores. *Linux magazine*, (86), 65–68. **retrieved from** https://dialnet.unirioja.es/servlet/articulo?codigo=4037882

Python.org. (yearmonthday). PEP 373 – python 2.7 release schedule [Python.org]. **retrieved** urlyearurlmonthurlday, **from** https://www.python.org/dev/peps/pep-0373/

Qian, Y., Hambrusch, S., Yadav, A., Gretter, S., & Li, Y. (yearmonthday). Teachers' perceptions of student misconceptions in introductory programming. *Journal of Educational Computing Research*, 0735633119845413. doi:10.1177/0735633119845413

Rubio, M. Á. & González del Valle, F. (yearmonthday). Uso de una herramienta de corrección automática en un curso de programación:

Una experiencia docente. *Actas de las Jornadas sobre Enseñanza Universitaria de la Informática*, *3*, 345–350. **retrieved from** http://www.aenui.net/ojs/index.php?journal=actas_jenui&page=article&op=view&path%5B%5D=437

TIOBE Software. (yearmonthday). TIOBE programming community index, may 2019. **retrieved from** https://www.tiobe.com/tiobe-index/

Vitale, B. (yearmonthday). A psycopedagogical approach to teaching of a programming language in secondary schools. *Infancia y Aprendizaje: Journal for the Study of Education and Development*, (50), 63. **retrieved from** https://dialnet.unirioja.es/servlet/articulo?codigo=48350

Páginas de consulta sobre licencias, legislación y desarrollo y uso del software analizado

[1] `Creative Commons Share Alike 4.0`. https://creativecommons.org/licenses/by-sa/4.0/

[2] `Diccionario RAE`. http://dle.rae.es/

[3] `Boletín Oficial del Estado`. https://www.boe.es/

[4] `Todo FP`. https://www.todofp.es/

[5] `ADIDE`. https://www.adideandalucia.es/

[6] Wikibooks (`LaTeX`). https://en.wikibooks.org/wiki/LaTeX

[7] `Code Runner`. https://github.com/trampgeek/moodle-qtype_coderunner/

[8] `Ejemplos del Hola Mundo`. https://es.wikipedia.org/wiki/Anexo:Ejemplos_de_implementaci%C3%B3n_del_%C2%ABHola_mundo%C2%BB

Otro material

- Diversas consultas puntuales al sitio `Stack OverFlow`.
- Material docente de las asignaturas **Procesos y contextos educativos**, **Innovación docente e Investigación Educativa en Ciencia y Tecnología** y **Complementos de Formación** impartidas en el Máster en Profesorado de Enseñanza Secundaria

Obligatoria y Bachillerato, Formación Profesional y Enseñanzas de Idiomasde la **Universidad de Granada**.