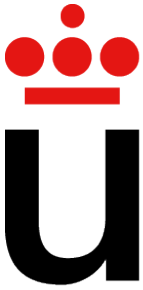

Universidad
Rey Juan Carlos

GRADO EN INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

Curso Académico 2020/2021

Trabajo Fin de Grado

IDENTIFICANDO EL NIVEL DE CÓDIGO PYTHON UTILIZANDO EL MARCO CERFL COMO INSPIRACIÓN

Autor : Ana Poveda García Herrero

Tutor : Dr. Gregorio Robles Martínez

Trabajo Fin de Grado

Identificando el nivel de código Python utilizando el marco CEFRL como inspiración

Autor : Ana Poveda García Herrero

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2021, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2021

Los científicos estudian el mundo tal como es;
los ingenieros crean el mundo que nunca ha sido.
Theodore Von Karman

II

Agradecimientos

Este proyecto pone fin a cinco años de una etapa dura, pero bonita. Porque ha habido momentos buenos y no tan buenos, pero siempre sacando la parte positiva para crecer como persona.

En primer lugar, agradecer a mi familia por siempre creer en mí, en especial a mis padres. A mi padre, por guiarme por este camino que por fin he conseguido, a mi madre, por transmitirme siempre su serenidad en momentos que las cosas no salían como yo quería. Porque siempre sacan lo mejor de mi, sin ellos no hubiera sido posible conseguirlo. También destacar a mi hermano, el cual es mi ejemplo a seguir, siempre ha confiado plenamente en mi, ayudándome en cada momento que lo he necesitado.

Por otro lado, quiero dar las gracias a mis amigas; ha sido una etapa dura de estudio en la que no he podido disfrutar al máximo con ellas, pero siempre han estado para animarme.

Gracias también a todos los compañeros de la universidad que me han acompañado en estos años, por hacer piña y ayudarnos siempre. En especial a mi amiga Tina; me llevo una gran amiga.

También de la universidad me llevo a mi novio, gracias por tu eterna paciencia y la ayuda infinita.

Por último, dar las gracias a todos los profesores que han hecho posible que llegue hasta aquí, cada uno ha puesto su granito de arena para que pueda llegar a decir *'I'm an engineer'*. Agradecer especialmente a mi profesor de este trabajo, Gregorio Robles, gracias por tu dedicación diaria, creer en mí y sobre todo transmitirme tu pasión por *Python*.

Resumen

El objetivo de este Trabajo de Fin de Grado es implementar una herramienta capaz de realizar un análisis de código en el lenguaje Python, con el fin de obtener su nivel de complejidad. Para ello, nos hemos inspirado en los niveles del CEFRL, el Common European Framework of Reference for Languages, que es ampliamente utilizado para lenguajes naturales a día de hoy en Europa. Se pretende que sea una herramienta utilizada en diferentes ámbitos, pero sobre todo en el ámbito educativo para poder obtener la evolución de cada alumno mediante el análisis de sus prácticas, el código que se le puede presentar, entre otros.

Para llevar a cabo este proyecto se han utilizado varias tecnologías. La principal ha sido el propio lenguaje de programación Python para desarrollar el programa, JSON utilizado como almacenamiento de datos, GitHub para obtener los repositorios a analizar y finalmente JavaScript, HTML y CSS para mostrar el análisis de forma más visual.

Summary

The goal of this Final Degree Project is to implement a tool capable of performing a code analysis in the Python language, in order to obtain its level of complexity. For this purpose, we have been inspired by the CEFRL levels, the Common European Framework of Reference for Languages, which is widely used for natural languages nowadays in Europe. It is intended to be a tool used in different fields, but especially in the educational field to obtain the evolution of each student through the analysis of their practices, the code that can be presented to them, among others.

Several technologies have been used to carry out this project. The main one has been the Python programming language itself to develop the program, JSON used as data storage, GitHub to obtain the repositories to analyze and finally JavaScript, HTML and CSS to display the analysis in a more visual way.

Índice general

Índice de figuras

Capítulo 1

Introducción

El Marco Común Europeo de Referencia para las lenguas (MCER o CEFRL) es el marco estándar en Europa, aunque es también ampliamente utilizado en otras regiones, que define la competencia lingüística. Se utiliza para definir las destrezas lingüísticas de los estudiantes en una de escala de seis niveles de referencia (A1, A2, B1, B2, C1 y C2), siendo A1 el nivel básico y C2 para aquéllos que dominen el idioma. Así, según se vaya avanzando en el conocimiento y la práctica del idioma, se obtiene un nivel superior, como se puede observar en la figura 1.1.

Este marco se ha extendido mucho en los últimos años en la sociedad, ya que en el mundo laboral te exigen tener ciertas competencias para poder desempeñar un puesto de trabajo. Una de estas competencias es el nivel que tienes en un idioma que puede ser inglés, español, alemán, chino, entre otros. Así, es común encontrar en el *currículum vítae* de los candidatos su nivel de destreza en un idioma siguiendo la convención del CEFRL.

Centrándonos en puestos de trabajo más tecnológicos, como puede ser el ámbito de la informática o de las telecomunicaciones, un requerimiento es saber programar en ciertos lenguajes y con un determinado nivel para poder realizar tu proyecto correctamente. A veces no se puede decir con certeza qué nivel se tiene en cada lenguaje de programación, ya que no hay muchas herramientas para evaluarte. Sería, por tanto, muy interesante contar con algo parecido al CEFRL para lenguajes de programación.

Esta última idea, la de aplicar CEFRL a habilidades de programación, no es nuestra. Ya en [8] se desarrolla una tabla que caracteriza el nivel de competencia que programadores han de tener en el contexto de varias actividades de programación. Este modelo se inspira en la tabla CEFRL, dividiéndose en tres niveles: “Usuario básico” (A), “Usuario independiente” (B)

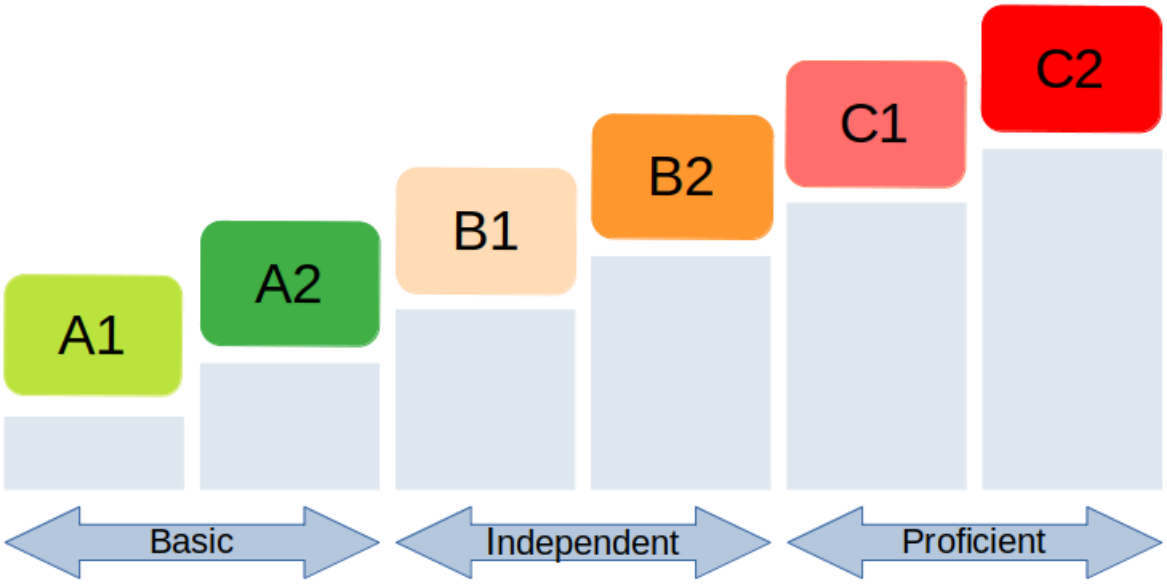

Figura 1.1: Esquema del CEFRL.

y “Usuario competente” (C).

Aunque con un enfoque ligeramente diferente, en este proyecto nos planteamos crear un esquema de evaluación similar al CEFRL, en particular para el lenguaje de programación Python. La idea surgió a partir de la curiosidad de saber qué nivel se tiene en este lenguaje, o sea, saber qué ‘tan bueno’ eres programando en Python. Además de la parte teórica, que se ha intentado evaluar con desarrolladores de Python, se ha desarrollado una herramienta software que permite evaluar los repositorios siguiendo el esquema que hemos elaborado.

Nótese que esta herramienta se plantea que sea útil también para el ámbito educativo, tanto para profesores que puedan evaluar el nivel en este lenguaje, como para los propios estudiantes que pueden ver su evolución a lo largo del curso. Podría a su vez ser interesante para catalogar repositorios o incluso trozos de código. Así, se podrían etiquetar las respuestas en StackOverflow según su complejidad, porque aunque el lema de Python es “sólo hay una manera de hacer las cosas”, esto no es del todo cierto; hay varias maneras, algunas más pitónicas que otras [3]. Y lo que pasa es que cuanto más pitónico, más avanzado es el nivel requerido por el desarrollador, haciendo que muchas veces se copien trozos de código sin entenderlo completamente, con los riesgos que eso supone.

1.1. Estructura de la memoria

La memoria de este trabajo se divide en los siguientes capítulos, resumidos a continuación:

- **Capítulo 1: Introducción.** Se explica el contexto en el que se ha desarrollado, su motivación y la estructura de la memoria.
- **Capítulo 2: Objetivos.** Se muestran tanto el objetivo principal como los objetivos específicos que lo componen y una planificación temporal del proyecto.
- **Capítulo 3: Estado del arte.** Se explica detalladamente las tecnologías usadas en el desarrollo del proyecto.
- **Capítulo 4: Diseño e implementación.** Se explica detalladamente cómo se ha diseñado y desarrollado la herramienta de evaluación de código Python.
- **Capítulo 5: Marco de aplicación.** Se muestra el contexto del proyecto y los resultados de las encuestas que hemos creado para desarrolladores con la finalidad de validar nuestro modelo.
- **Capítulo 6: Resultados y experimentos.** Se incluyen las pruebas realizadas al respecto con usuarios de GitHub y sus resultados.
- **Capítulo 7: Conclusiones.** Se muestran las conclusiones finales del proyecto, objetivos alcanzados y una breve valoración personal. Por último, se presentan posibles líneas futuras que se podrían desarrollar para mejorar la herramienta.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo de este Trabajo de Fin de Grado es crear una herramienta capaz de obtener una evaluación inspirada en el "Common European Framework of Reference for Lenguajes" para código escrito en el lenguaje de programación Python, versión 3.

A través de esta herramienta, se podrá analizar el nivel de los repositorios (y sus desarrolladores) de GitHub o de trozos de código en este lenguaje.

2.2. Objetivos específicos

Para alcanzar el objetivo principal se han planteado los siguientes objetivos específicos:

- Analizar y extraer todos los elementos que componen el lenguaje de Python a través de libros de este lenguaje. Este punto es primordial para poder elaborar una primera versión de niveles, CEFRL.

- Crear una primera versión del programa que analice directorios de ficheros *.py*, creando una base de datos con los elementos encontrados con su localización y nivel correspondiente. Tras esto, hacer una mejora analizando usuarios de GitHub.

- Crear un fichero de configuración para que cada usuario que utilice el programa pueda establecer sus propios niveles.

- Mostrar los resultados de una manera más visual mediante páginas web.

- Verificar el uso de CEFRL con elementos de Python. Esto se hará mediante encuestas para obtener una valoración de nuestra propuesta. Con los resultados obtenidos a partir de desarrolladores de Python, pretendemos mejorar el programa y calcular la fiabilidad de nuestro modelo.

- Probar la herramienta con los repositorios de la asignatura *'ptavi'* (Protocolos para la Transmisión de Audio y Vídeo de Internet, del tercer curso del Grado en Ingeniería en Sistemas Audiovisuales y Multimedia de la ETSIT-URJC) para obtener resultados de alumnos.

- Probar la herramienta con otros repositorios de GitHub para obtener resultados con la herramienta y poder valorar su fiabilidad.

2.3. Planificación temporal

La idea del proyecto nació en octubre de 2020, al plantearse la idea inicial y sus objetivos. Tras esto, me dispuse a documentarme con libros de Python para poder extraer los elementos del lenguaje, pero al poco tiempo empecé las prácticas de la carrera más alguna asignatura que me quedaba. De esta manera, dediqué el mayor tiempo del proyecto en algunas mañanas y sobre todo los fines de semana, ya que por las tardes trabajaba.

En febrero de 2021, empecé a dedicarle más tiempo al proyecto ya que lo había tenido un poco apartado, y empecé a indagar sobre el módulo *ast* para crear árboles de sintaxis abstracta de cada código y extraer así los diferentes atributos disponibles.

A partir de marzo, terminé las prácticas y pude dedicarme al proyecto por las tardes. Se creó una primera versión de niveles basada en el CEFRL. A estas alturas la herramienta ya guardaba los elementos extraídos y les asignaba un nivel. Los resultados se ofrecían en formato *.csv* y *.json* para poder visualizarlos mejor.

En abril se decide hacer un fichero de configuración *.cfg* para que cada usuario pueda configurar el nivel considerado a la clase deseada. A partir de este fichero se crea un diccionario donde están guardados los niveles para el programa principal. A mediados de este mes, se crea

una página web sencilla para visualizar los datos obtenidos en los análisis y, además, se elaboran dos formularios para realizar encuestas a desarrolladores con la finalidad de verificar los niveles establecidos para cada clase.

En mayo se comienza a realizar la memoria. A mediados, se lanzan las encuestas, para que lleguen a los primeros desarrolladores, y así poder probarlas. Tras probarlos con varios profesores y alumnos de la ETSIT-URJC, a principios de junio se comparten las encuestas finales por las redes sociales para que llegue al máximo número de desarrolladores de Python.

Tras unas semanas, se recogen los resultados obtenidos y se hace un análisis de ellos para obtener una validación final de nuestra herramienta. Finalmente, a principios de julio se entrega el proyecto y se prevé su defensa en fechas próximas.

Capítulo 3

Estado del arte

En este apartado se describen las diferentes tecnologías que se han utilizado para llevar a cabo este proyecto.

3.1. Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Fue creado por Guido Rossum a principios de los años noventa.

Se trata de un lenguaje de código abierto cuyas principales características son:

- Lenguaje multiparadigma. Permite diferentes paradigmas como son la programación orientación a objetos, la programación imperativa y la programación funcional.
- Tipado dinámico. No hay una declaración del tipo de variable hasta la ejecución.
- Fuertemente tipado. No se permite tratar a una variable de un tipo distinto al suyo.
- Multiplataforma. Soporta diferentes plataformas como Linux, Windows o MAC.

Una de las fortalezas de Python, quizás la mayor, es la biblioteca estándar que posee, con decenas de módulos que cubren la mayoría de las necesidades básicas de un programador.

Por último, destacar la filosofía de Python, la cual tiene unos principios que al cumplirlos el código se considera "pitónico", entre los que se encuentran:

- Bello es mejor que feo.

- Simple es mejor que complejo.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.

Aunque el Zen de Python[1] dice que "There should be one– and preferably only one –obvious way to do it." (debería haber una– y preferentementne solo una –manera obvia de hacerlo), las soluciones en Python se pueden plasmar de diferentes maneras. En este TFG se aboga porque a veces esa manera obvia es muy compleja, al menos para un iniciado, y que podría ser más interesante mostrar código que sea entendido.

3.2. JavaScript

JavaScript[2] es un lenguaje de programación ligero e interpretado, y más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web. Además, es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat.

Es un lenguaje de programación basado en prototipos, dinámico, levemente tipado y multi-paradigma con soporte para programación orientada a objetos, imperativos y funcionales.

Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas y JavaScript del lado del servidor (Server-side JavaScript o SSJS).

3.3. HTML5

HTML5 (HyperText Markup Language)[3] es la quinta versión y última de HTML.

Es un lenguaje de marcado de páginas web, que define una estructura básica para la definición de contenido de una página web. HTML5 tiene una estructura de árbol de elementos y texto. Cada elemento se denota en el código con etiquetas de inicio, junto con atributos, y etiquetas de fin. El contenido se encuentra entre las dos etiquetas.

[1] `https://www.python.org/dev/peps/pep-0020/`
[2] `https://developer.mozilla.org/es/docs/Web/JavaScript`
[3] `https://developer.mozilla.org/es/docs/Web/Guide/HTML/HTML5`

Se complementa con otras tecnologías como CSS, para modificar la apariencia de la página o JavaScript, para determinar la funcionalidad.

3.4. CSS3

CSS3 (Cascading Style Sheets) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Este lenguaje está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características como las capas o layouts, los colores y las fuentes. Esta separación, hace posible presentar el mismo documento con distintos estilos.

Junto con HTML y JavaScript, CSS es una tecnología usada por muchos sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web y GUIs para muchas aplicaciones móviles.

El encargado de definir y mantener las especificaciones de este lenguaje es World Wide Web Consortium (W3C).

3.5. JSON

JavaScript Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es un formato ligero para el intercambio de datos entre aplicaciones.

JSON esta pensado para ser un valor pasado como parámetro a una función o bien el valor devuelto por una función, por lo que el archivo json está formado por un único elemento. Este elemento puede ser bien un elemento simple o un elemento compuesto.

- Elementos simples:
    - null.
    - true.
    - false.
    - Cadenas. Cualquier valor situado entre comillas.
    - Números. Expresados en base 10.

- Elementos compuestos. Son elementos que contienen otros elementos a su vez:
  - Objeto. Contiene una colección de pares Nombre/valor.
  - Array. Una lista ordenada de elementos.

3.6. GitHub

GitHub es un servicio de alojamiento de repositorios utilizando el sistema de control de versiones Git[4], con el que los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las nuevas versiones que sacan de sus aplicaciones para evitar confusiones.

El código de los proyectos alojados en GitHub se almacena típicamente de forma pública. Además, ofrece herramientas propias como las siguientes:

- Wiki para cada proyecto. Ofrecer toda la información sobre el proyecto y anotar todos los cambios de las diferentes versiones.
- Sistema de seguimiento de problemas. Otras personas pueden hacer mejoras, sugerencias y optimizaciones en los proyectos
- Revisión de código. Se pueden dejar anotaciones para que su creador las pueda revisar
- Gráficos de trabajo. Visualizar la forma de trabajo en proyectos y sus bifurcaciones, partir de los cambios realizados a través de “commits”.

3.7. Proyecto de programación del CEFRL

Este proyecto de programación [8] está basado en una tabla que caracteriza el nivel de competencia de los programadores de un lenguaje de programación usando los niveles del CEFRL, con diferentes habilidades de programación. Esta tabla se puede ver en las figuras A.1, A.2 y A.3 en la sección A.1 del anexo A.

Como se puede comprobar, caracteriza los niveles en tres competencias, y cada una de ellas en otras derivadas:

[4] `https://cutt.ly/es`

1. **Writing.**
    - Writing code.
    - Refactoring.
    - Embedding in a larger system.
2. **Understanding.**
    - Reusing code.
    - Explaining/Discussing code.
3. **Interacting.**
    - Exploring, self-learning.
    - Mastery of the environment.
    - Troubleshooting.

Al igual que el CEFRL, esta tabla divide en tres los niveles: “Usuario básico” (A) para aquellos que pueden realizar la tarea bajo supervisión o con una guía; “Usuario independiente” (B) para aquellos que pueden realizar la tarea sin supervisión pero con alguna orientación; y “Usuario competente” (C) para aquellos que sean totalmente independientes. Y estos tres niveles, se dividen cada uno en dos subniveles (A1, A2, B1, B2, C1, C2).

Esta tabla se puede usar de diferentes maneras dependiendo que quieras conseguir, por ejemplo:

- Evaluar el nivel por habilidad.
- Determinar la habilidad más desarrollada
- Evaluar el nivel mínimo para un lenguaje de programación.

Además, para poder usar este método de evaluación, hay creada una aplicación[5] en la cuál puedes elegir el lenguaje de programación a evaluar como se observa en la figura 3.1

Tras esto, te van evaluando teóricamente según lo que tú consideres que eres capaz de hacer, un ejemplo se ve en la figura 3.2. Cuando llega al final, te indica el porcentaje de cada habilidad en un nivel.

[5]https://dr-knz.net/programming-levels/test

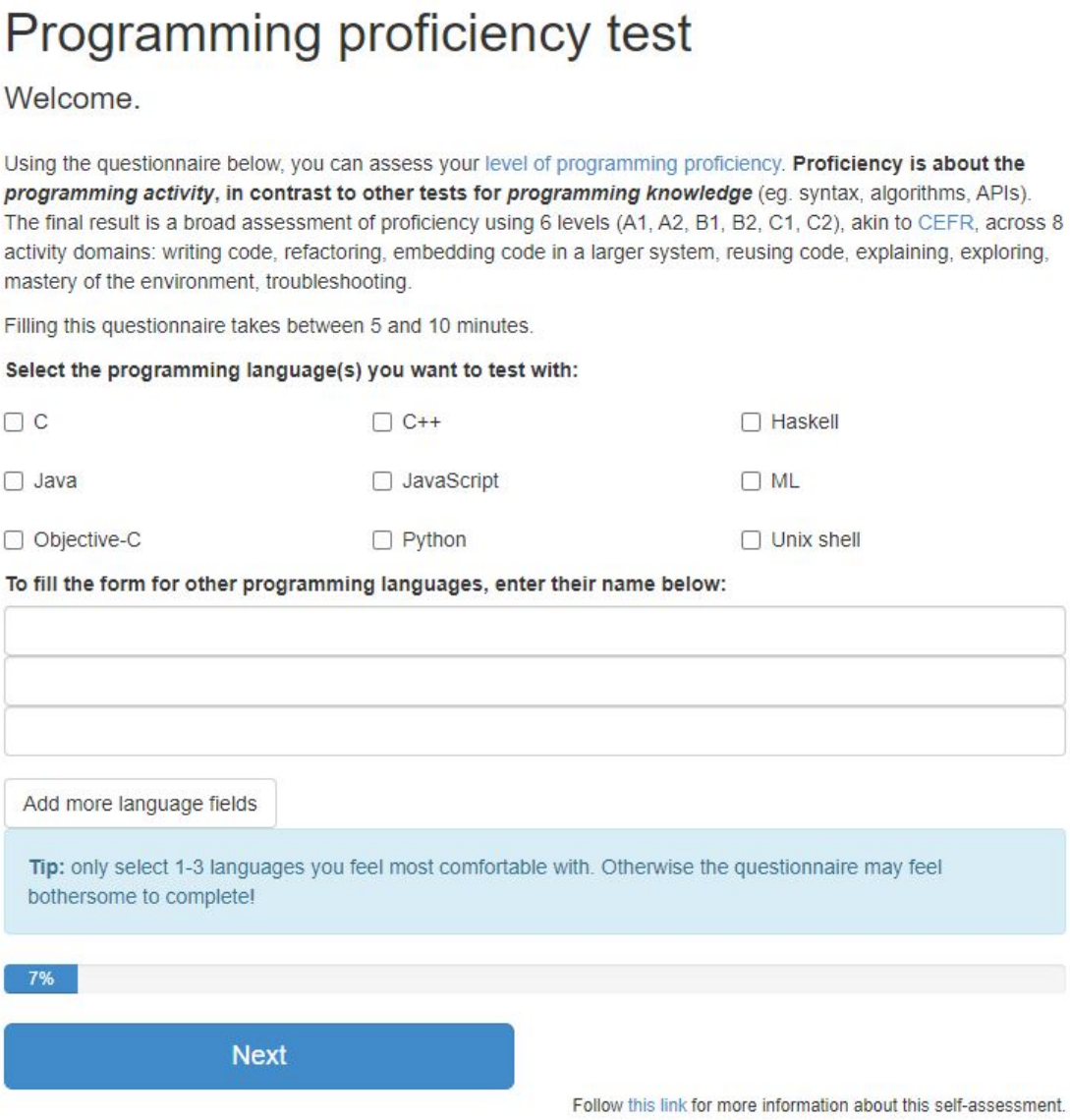

Figura 3.1: Aplicación del proyecto de programación CEFRL.

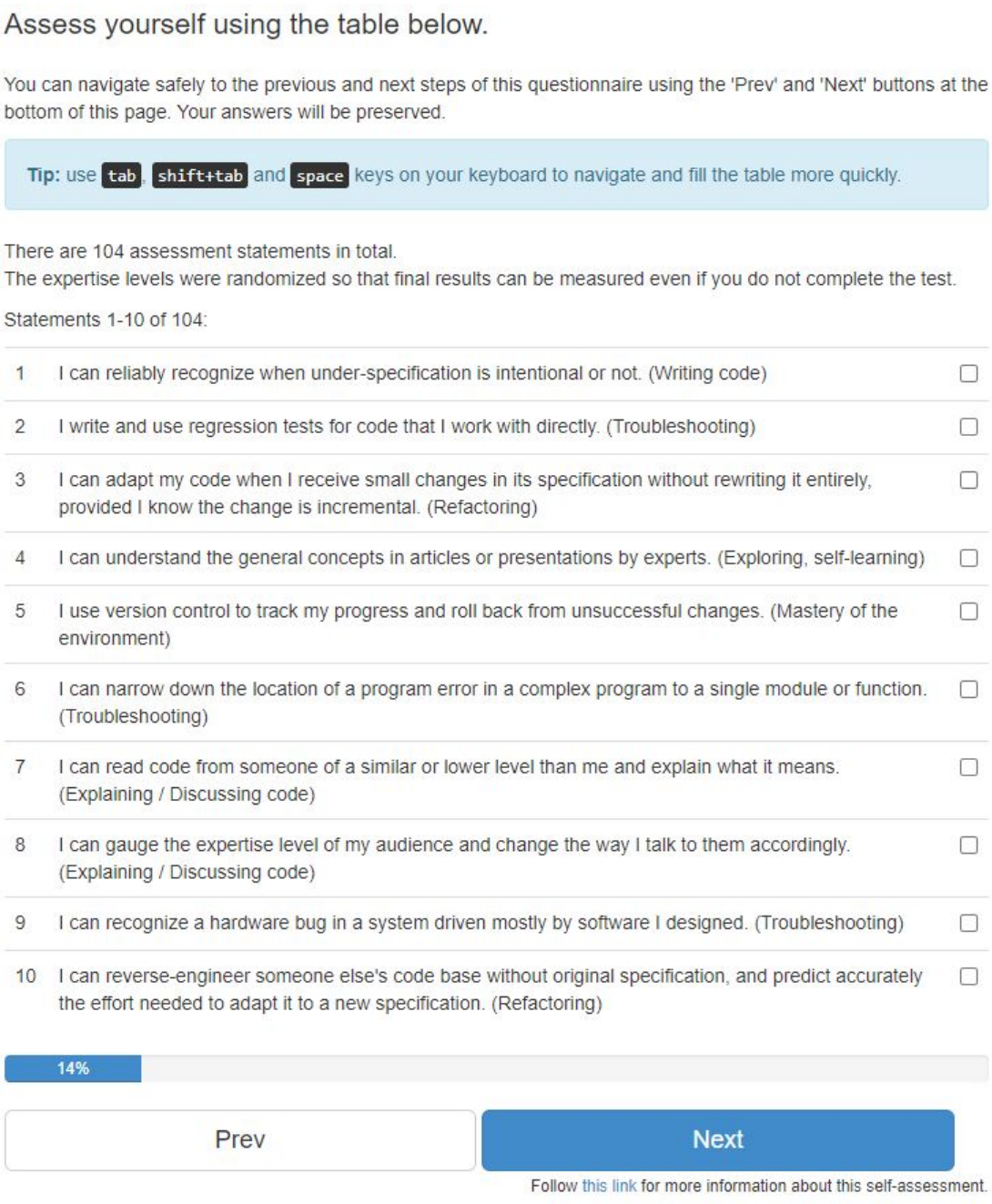

Figura 3.2: Cuestiones de la aplicación del proyecto de programación CEFRL.

3.8. Jupyter Notebook

Jupyter Notebook [1] es una aplicación web de código abierto que le permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Suele ser usado junto con la biblioteca *pandas*. Sus principales usos son los siguientes:

- **Depuración de datos**. Distinguir entre los datos que son importantes y los que no lo son al ejecutar un análisis de big data.
- **Modelado estadístico**. Estimación de probabilidades de distribución de ciertas características.
- **Visualización de datos**. Mediante el uso de bibliotecas como *Numpy*[6] y *Matplotlib*[7] se pueden crear y visualizar representaciones gráficas.
- **Creación de modelos de aprendizaje automático**.

3.9. Pandas

Pandas [2] es una biblioteca de código abierto que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento y fáciles de usar para el lenguaje de programación Python.

Tiene numerosas características esta biblioteca, como pueden ser la creación de nuevos arrays con la biblioteca *Numpy*, el acceso de datos mediante índices, la lectura y escritura de ficheros en formato Excel, CSV, entre otros.

3.10. Coeficiente de Kappa de Cohen

El coeficiente kappa (k) de Cohen [4] es una herramienta estadística diseñada para conocer el grado de concordancia entre dos observadores, es decir, hasta qué punto ambos coinciden en su medición.

[6] https://numpy.org/

[7] https://matplotlib.org/

Coefficient Kappa	Strength of concordance
0,00	Poor
0,01 – 0,20	Slight
0,21 – 0,40	Fair
0,41 – 0,60	Moderate
0,61 – 0,80	Substantial
0,81 – 1,00	Almost perfect

Figura 3.3: Valoración del coeficiente de kappa [5].

El coeficiente kappa toma valores entre -1 y +1; mientras más cercano a +1, mayor es el grado de concordancia inter-observador. Por el contrario, un valor de k = 0 refleja que la concordancia observada es precisamente la que se espera a causa exclusivamente del azar.

La interpretación del coeficiente kappa se realiza relacionando su valor con una escala cualitativa que incluye seis niveles de fuerza de concordancia como se observa en la figura 3.3.

La ecuación para *k* es:

$k = \frac{Pr(a)-Pr(e)}{1-Pr(e)}$

siendo:

- Pr(a) es el acuerdo observado relativo entre los observadores.
- Pr(e) es la probabilidad hipotética de acuerdo por azar.

En Python, el coeficiente de kappa de Cohen esta implementado en una función llamada *'cohen_kappa_score'*[8] contenido en el módulo *'skelearn.metric'*.

3.11. Coeficiente de Kappa de Fleiss

El coeficiente kappa (k) de Fleiss [7] es una generalización del *coeficiente de kappa de Cohen* de manera que mide el acuerdo entre más de dos observadores. Este coeficiente, añade el cálculo del sesgo del codificador (precisión-error) y el cálculo de la concordancia (calibración). La fórmula es la siguiente:

$$\overline{K} = 1 - \frac{n \cdot m^2 - \sum_{i=1}^{n} \sum_{j=1}^{r} x_{ij}^2}{n \cdot m \cdot (m-1) \cdot \sum_{j=1}^{r} \overline{p_j} \cdot \overline{q_j}} \tag{3.1}$$

[8]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html

siendo:

- n: número total de conductas o códigos a registrar.
- m: número de codificaciones.
- xij: número de registros de la conducta i en la categoría j.
- r: número de categorías de que se compone el sistema nominal.
- p: es la proporción de acuerdos positivos entre codificadores.
- q: es la proporción de acuerdos negativos (no acuerdos) en codificadores (1 - p).

En Python, el coeficiente de kappa de Fleiss esta implementado en una función llamada *'multi_kappa'* contenido en el módulo *'nltk.agreement'*[9].

[9]https://www.nltk.org/_modules/nltk/metrics/agreement.html

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

La herramienta implementada en este proyecto consta de diferentes pasos como ya vimos en la sección 2.2. Para que quede más claro, observamos la figura 4.1, en la cual se puede apreciar un diagrama de funcionamiento. Este diagrama consta de un *programa principal* en Python, el cual recibe *parámetros* por la *shell* que puede ser un *directorio* o un *repositorio* o *usuario* de GitHub. En el caso de que sea un parámetro de GitHub, se hará uso de su API para extraer los ficheros con extensión .py a analizar. Además, destaca la clase *IterTree* que, mediante el módulo de ast (*Árboles de Sintaxis Abstracta*) y un fichero de configuración *.cfg* para establecer el CEFRL de cada elemento, va a extraer los niveles y clases de los elementos que componen los ficheros con extensión .py a analizar.

Finalmente, el programa principal va a devolver los resultados en formato *CSV* y *JSON*. Este último formato se va a utilizar como una especie de *base de datos* sencilla para la creación de una página web mediante HTML, JavaScript y CSS para que sea más visual para un *humano*.

4.2. Configuración del fichero CEFRL

Se crea un fichero de configuración *.cfg* llamado ***'configuration.cfg'***, para hacer personalizada la asignación de un nivel CEFRL a un elemento. Para ello se sigue la siguiente sintaxis:

```
[Sección1]
opción1 = valor1
```

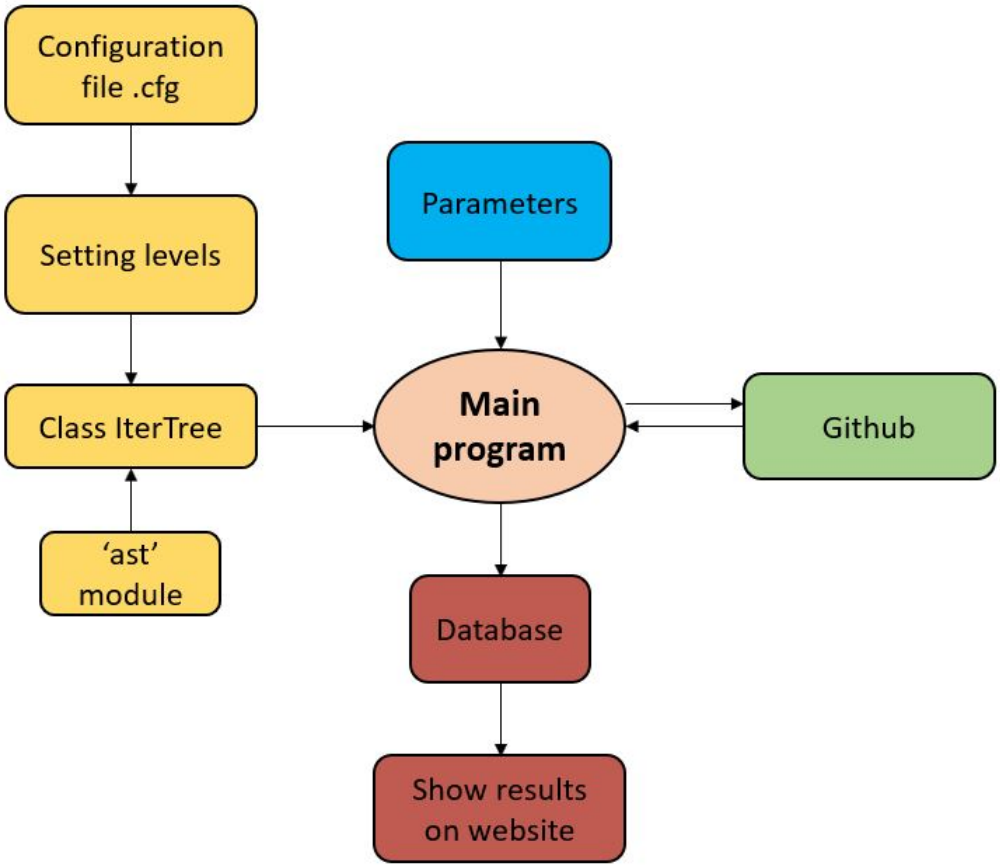

Figura 4.1: Esquema de funcionamiento.

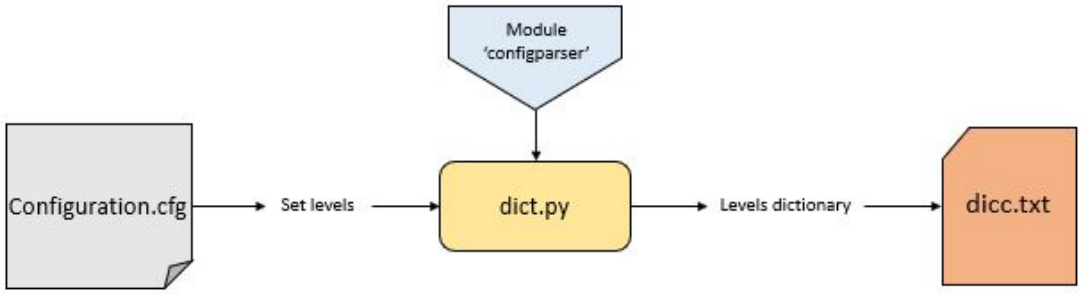

Figura 4.2: Esquema de la creación del diccionario de niveles.

```
opción2 = valor2

[Sección2]
opción1 = valor1
opción2 = valor2
```

siendo las *secciones* el elemento global, las *opciones* variaciones del elemento y los *valores* el nivel CEFRL (A1, A2, B2, C1, C2). A partir de este fichero de configuración, hacemos uso del módulo *'configparser'* para crear un nuevo programa ***'dict.py'***, que nos lee el fichero y forma un fichero de texto de nombre (***'dicc.txt'***) y que contiene un diccionario de niveles. A partir del mismo, se extraerán los valores, como se puede observar en la figura 4.2.

4.3. Obtención de ficheros a analizar

En este punto, se reciben los parámetros por la *shell* de manera que dependiendo de lo introducido obtendrá más o menos ficheros a analizar y procedente de una ubicación u otra. Para ejecutar entonces nuestro programa principal, se debe ejecutar lo siguiente:

```
python3 main.py opción valor
```

Por lo tanto, se tiene tres opciones e infinitos valores. Las posibles opciones son las siguientes:

1. **'directory'**. Haciendo referencia a la definición de un directorio [1], un directorio es un contenedor virtual en el que se almacenan una agrupación de archivos informáticos y otros subdirectorios, atendiendo a su contenido, a su propósito o a cualquier criterio que decida el usuario. De esta manera, el valor a introducir debe ser una ruta hacía la carpeta deseada a analizar. Por ejemplo:

   ```
   /home/ana/Documentos/TFG
   ```

2. **'repo-url'**: En este caso, hay que introducir la url de un repositorio de GitHub. Para conseguir la url de un repositorio, hay que dirigirse a GitHub, meternos en el repositorio que deseemos analizar, dar al botón 'clone' y copiar la url de *https*. De esta manera, la URL será nuestro valor. Un ejemplo de URL sería de la siguiente manera:

   ```
   https://github.com/anapgh/ptavi-p4.git
   ```

3. **'user'**: En esta última opción, hay que introducir como valor el nombre de usuario de GitHub que se desea analizar. Por lo tanto, tendrá para analizar los diferentes repositorios que tenga el usuario indicado.

Después de introducir los parámetros opción y valor, como solo se quiere analizar los ficheros programados en Python, se seleccionarán solo los ficheros con extensión *.py*. En el caso de las dos últimas opciones, al hacer uso de la herramienta GitHub la extracción de ficheros es más compleja.
Para ello, hay que hacer solicitudes GET a la API de GitHub, usando el modulo 'requests'. Esta API, contiene toda la información en formato JSON. Se extrae primeramente la información de los lenguajes usados en cada repositorio, para comprobar si el repositorio tiene al menos un 50 % de Python, y solo si es así, se clona dicho repositorio en el cual solo se analizarán los archivos con extensión *.py*.

[1] https://www.hostgator.mx/blog/que-es-un-directorio/

4.4. Iteración sobre árboles de sintaxis abstracta

Tras obtener los ficheros *.py*, se va a hacer uso del módulo ***ast***[2]. Un punto importante a tratar es que este módulo solo puede manejar código Python para la versión de Python 3. Por lo que si se le pasa una estructura básica de Python 2, como puede ser el elemento *print* sin paréntesis, lanzaría la excepción *SyntaxError*, ya que en Python 3 se escribe diferente como se puede apreciar:

- **Python2**:

```
print hello
```

- **Python3**:

```
print(hello)
```

Esto es debido a que usa el mismo analizador para compilar el código de Python en bytes, siendo éste incompatible con versiones anteriores.

Con el módulo *ast* se va a generar un árbol de sintaxis abstracta usando el ayudante *parse()*. El resultado será un árbol de objetos cuyas clases todas heredan de *ast.AST*.

Para ello, primero se definen las clases que nos interesen analizar que harán referencia a los distintos elementos del lenguaje Python.

```
Literals = ['ast.List', 'ast.Tuple', 'ast.Dict']

Variables = ['ast.Name']

Expressions = ['ast.Call', 'ast.IfExp', 'ast.Attribute']

Comprehensions = ['ast.ListComp','ast.GeneratorExp',
                  'ast.DictComp']
```

[2]https://docs.python.org/es/3.10/library/ast.html

```
Statements = ['ast.Assign', 'ast.AugAssign', 'ast.Raise',
              'ast.Assert','ast.Pass']

Imports = ['ast.Import', 'ast.ImportFrom']

ControlFlow = ['ast.If', 'ast.For', 'ast.While',
               'ast.Break', 'ast.Continue', 'ast.Try',
               'ast.With']

FunctionsClass = ['ast.FunctionDef', 'ast.Lambda',
                  'ast.Return', 'ast.Yield',
                  'ast.ClassDef']
```

Con esto, va a aparecer la clase ***IterTree*** la cual va a ir creando objetos con el método constructor `__init__` compuestos por:

- **tree**. Es el árbol creado del repositorio/carpeta donde se encuentra.
- **attrib**. Es la clase del módulo ast.
- **file**. Es el nombre del fichero.py.
- **repo**. Es el nombre del repositorio/carpeta a analizar.

Vamos a ir recorriendo cada nodo del árbol con el método *ast.walk()* comprobando si el tipo de ese nodo corresponde a alguna de las clases del módulo ast que se ha definido. Si es así, se comprobará los elementos que tiene dentro de ese nodo mediante el programa ***levels.py***, de esta manera, según el elemento se asignará la clase correspondiente y el nivel asignado a esa clase. Este nivel se abstrae del diccionario de niveles creado en la sección 4.2. A continuación, teniendo la clase y el nivel de ese nodo, se crea un fichero en formato JSON y CSV con los siguientes datos:

- Nombre del repositorio.
- Nombre del fichero.

- Clase del elemento.
- Línea donde empieza.
- Línea donde acaba.
- Desplazamiento de columna.
- Nivel del elemento.

De esta manera, se obtienen los datos del análisis en los fichero *'data.json'* y *'data.csv'*.

4.5. Análisis de datos

A partir de los datos obtenidos en el análisis, nos vamos a centrar en el fichero JSON. De este fichero, vamos a obtener distintos tipos de análisis que se van a guardar en una carpeta llamada *DATA_JSON*.

4.5.1. Resumen individual de cada repositorio

Orientado a los repositorios, vamos a diferenciar tres tipos de archivos:

- **Archivo JSON por cada repositorio, *'name_repository.json'***. Dentro de este, nos vamos a encontrar como elemento raíz el nombre del repositorio, el cual va a derivar en distintos archivos.py. Estos archivos.py, van a tener cada uno dos claves, *'Levels'* con el numero total de cada nivel (A1, A2, B1, B2, C1, C2), y *'Class'* con los tipos de clases que aparecen en ese archivo y el numero de apariciones.
- **Archivo JSON con todos los repositorios, *'total_data.json'***. Aquí, se encuentran todos los repositorios, cada uno con la misma estructura del punto anterior.
- **Archivo JSON con resumen de los repositorios, *'repo_data.json'***. En él se encuentran todos los repositorios, pero a diferencia de los dos anteriores, en este caso no se trata de un análisis detallado de cada archivo con extensión .py del repositorio, sino que es un resumen de niveles y clases total.

Cada uno de estos archivos, se utilizarán para mostrar los resultados de forma ordenada en la página web.

4.5.2. Resumen general

Este análisis esta orientado a un resumen global de los niveles y clases de todos los repositorios. Este archivo, ***'summary_data'***, contiene dos únicas claves principales, que son *'Levels'* con el número total de cada nivel, y *'Class'* con el número total de los tipos de clases que se encuentran en todos los repositorios.

4.6. Elaboración de una página web

Después de haber creado los distintos ficheros de análisis, se decide crear una página web para que sea más visual para el *humano*. Para ello, se ha decidido usar **'Node.js'**[3] para que fuese más sencillo leer archivos y depurar errores. Esta página va a estar basada en cuatro archivos:

- main.js
- main.html
- repo.html
- main.css

El primer archivo va a ser un archivo JavaScript el cual va a realizar todas las funciones y análisis de esta página. Lo primero de todo, va a leer los ficheros en formato JSON que previamente se había creado, *total_data.json, repo_data.json, summary_data.json*.

A partir de estos ficheros, va a crear una página principal, ***index.html*** y páginas secundarias para cada repositorio, ***nombre_repositorio.html***. La página principal *index.html*, va a ser creada mediante *main.html*, en la cual se van a crear el mismo número de botones que repositorios se ha encontrado en los ficheros.json. Estos botones nos derivarán a las páginas personalizadas de cada repositorio.

En *index.html*, además de los botones que nos redireccionan a las páginas secundarias, podemos encontrar un resumen general del numero de niveles y de tipos de clases encontradas en todos los repositorios. Esta información, va a ser extraída de *summary_data.json*.

Las páginas secundarias, van a mostrar un resumen general del repositorio, *repo_data.json*, y un resumen detallado de cada archivo.py que compone el repositorio, *total_data.json*. Estas

[3] https://nodejs.org/es/

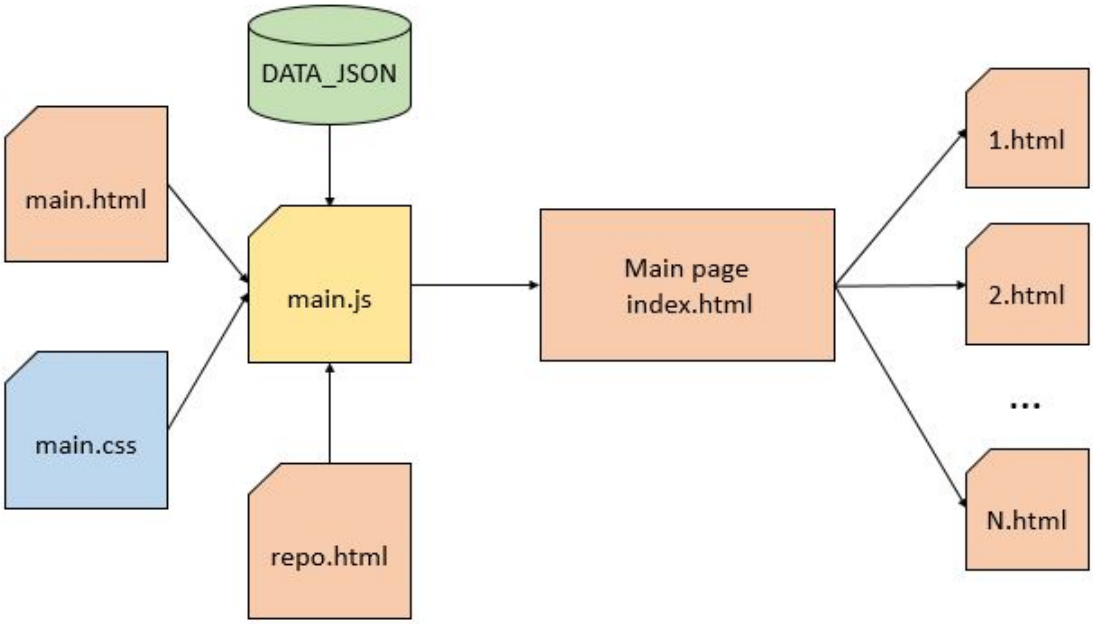

Figura 4.3: Esquema de la creación de la página web.

páginas van a ser creadas mediante un archivo plantilla llamado *repo.html*, que puede ser personalizado para cada repositorio. Además, el archivo *main.css* va a ser el encargado de dar estética a todas las páginas, como se puede observar en la figura 4.3.

Capítulo 5

Marco de aplicación

5.1. Comparativa de proyectos del CEFRL

Nuestro proyecto nace de la idea de conocer que nivel tienes en el lenguaje de Python versión 3. Como ya vimos en el capítulo 3, en la sección 3.7, hay una propuesta que tiene una idea parecida pero con varias diferencias. Esta herramienta que se ha creado, sirve para evaluar únicamente el lenguaje de Python, a diferencia de la propuesta anterior que permitía evaluar en otros lenguajes, como podría ser Java, C, JavaScript... Además, nuestro proyecto solo se centra en la habilidad escrita (*writing*), ya que su forma de evaluar es mediante código fuente escrito de Python. Esto último es otra diferencia; nuestro proyecto se basa en una herramienta que evalúa el código fuente, no en hacer preguntas teóricas. Por otro lado, al mostrar los resultados obtenidos, la propuesta anterior mostraba el porcentaje de cada nivel en una habilidad; en nuestro caso, al solo tener la habilidad de escritura, muestra la cantidad de elementos usados de un nivel en cada fichero, repositorio y usuario.

5.2. Listado elementos Python

En este proyecto se va a evaluar el tipo de elementos que se usan en un código escrito en Python3, ya que cada uno tiene un nivel diferente de complejidad. Para ello, el proceso de documentación de esta herramienta se centra en el libro *"Learning Python, 5th Edition"* [6], y se han extraído diferentes elementos a evaluar:

- **Estructuras básicas**: *List, Dictionary, Tuple*.

- **Estructuras más avanzadas**: *List Comprehension, Dict Comprehension.*
- **Elementos de ficheros**: llamadas como *'open', 'write', 'writelines', 'read' y 'readline'*.
- **Asignaciones**.
- **Sentencias de If**: *Simple If statements, If expression, using* `__name__`.
- **Bucles**: *While Loop, For Loop*.
- **Declaraciones de bucle**: *'break', 'continue', 'pass'*.
- **Funciones**: *Simple Function, Recursive Function.*
- **Llamadas**: *'return', 'print', 'lambda', 'range', 'enumerate', 'zip', 'map'*.
- **Importaciones**: *'import', 'from'*.
- **Generadores**: *Generator function, Expression function.*
- **Módulos**: como *'struct', 'pickle', 'shelve', 'dbm', 're', 'importlib'*.
- **Clases y elementos**: *Class, Class inherited,* `__init__`*, Descriptors, Properties, Private Methods and Attributes, staticmethod, classmethod.*
- **Decoradores**: *Decorator function, Decorator class.*
- **Metaclases**.
- **Función 'Super'**.
- **Excepciones**: *'try-except', 'try-else-except', 'try-finally', 'try-except-finally', 'raise', 'assert'*
- **Atributos**: *Simple attribute,* `__class__`, `__dict__`, `__slots__`.
- **With**

De algunos de estos elementos, se sacaron variantes como pueden ser *Lista de Listas, Dict comprehension anidados, Funciones con argumentos pasados por defecto, etc*. Con esto obtuvimos una lista de elementos de la sintaxis de Python3.

5.3. Propuesta de niveles

A partir de los elementos anteriores, basándonos en lecturas y en el nivel de Python propio, se propuso una primera versión de niveles de cada elementos usando como referencia los niveles del CEFRL. Esta primera versión de niveles se puede observar en las figuras A.4, A.5 y A.6 en la sección A.2 del anexo A.

Más tarde, se elaboró una estructura para poder clasificar de forma más lógica los elementos:

- **A1**: Nivel básico de estructuras de datos, flujo y lectura ficheros.
- **A2**: Estructuras de datos complejas, excepciones sencillas, funciones.
- **B1**: Orientación a objetos y estructuras avanzadas.
- **B2**: Estructuras pitónicas sencillas.
- **C1**: Estructuras pitónicas avanzadas.
- **C2**: Metaclases.

Entonces se elaboró una segunda versión de niveles a partir del nivel del tutor del proyecto. Esta versión se puede observar en las figuras A.7, A.8 y A.9 en la sección A.3 del anexo A.

Con estas dos versiones se decide calcular el ***coeficiente de kappa de Cohen*** para visualizar la diferencia entre ambas. Para ello, se hace uso de *Jupyter Notebook* con los módulos:

- **pandas**: Para realizar la lectura y la manipulación de los ficheros en formato CSV.
- **skelearn.metric**: Para obtener la función del cálculo del coeficiente de kappa llamada *cohen_kappa_score*.
- **numpy** y **pylab**: Para obtener las gráficas.

De esta manera, se extrae la columna de niveles de cada versión y se le asigna un valor del 1 a 6 haciendo referencia de A1 a C2. Esto lo comprobamos en la figura 5.1. Por lo tanto, en esta comparativa se obtiene un coeficiente de kappa de *0.258*. La gráfica de estas diferencias se puede observar en la figura 5.2.

Tras esto, nos damos cuenta que al desconocer un elemento, lo divergemos de manera que le establecemos un nivel mayor al nuestro. De esta manera, teniendo en el caso de la autora del

```
LEVELS:

First version:  ['A2', 'B1', 'C1', 'C2', 'C1', 'A2', 'B1', 'B1', 'B2', 'C1', 'C2', 'C2', '
C2', 'A1', 'A2', 'A2', 'A2', 'A2', 'A2', 'A2', 'A1', 'A1', 'A1', 'A2', 'A1', 'B1', 'B2', '
B1', 'B1', 'B1', 'A1', 'A2', 'A1', 'A2', 'A2', 'A2', 'A2', 'A2', 'C2', 'C2', 'C2', 'A1', '
A2', 'B1', 'B1', 'B1', 'B2', 'A1', 'B1', 'C1', 'C1', 'A2', 'A2', 'B1', 'B1', 'B1', 'B1', '
B1', 'B1', 'C1', 'C1', 'C2', 'C1', 'C1', 'C2', 'C2', 'C2', 'C2', 'C2', 'C2', 'C1', 'A2', '
B2', 'B2', 'B1', 'B1', 'B1', 'B2', 'B2', 'B2', 'B1', 'B1', 'B1']

Second version: ['A2', 'A1', 'B2', 'C1', 'B2', 'A1', 'A2', 'A2', 'B1', 'B2', 'B2', 'B2', '
C1', 'A1', 'A2', 'A1', 'A1', 'A1', 'A1', 'A1', 'A1', 'A1', 'A1', 'A2', 'A1', 'A1', 'A2', '
A2', 'A2', 'B1', 'A1', 'B1', 'A1', 'A2', 'A1', 'A1', 'A1', 'A2', 'B2', 'B2', 'B1', 'A2', '
B1', 'B2', 'B2', 'B2', 'B1', 'A2', 'C1', 'C1', 'C1', 'A2', 'A2', 'A2', 'A2', 'A2', 'B1', '
B1', 'B1', 'B2', 'B2', 'B2', 'C1', 'C1', 'C1', 'C2', 'C2', 'C2', 'C2', 'B2', 'C1', 'B1', '
B2', 'B2', 'A2', 'B1', 'A2', 'B1', 'B1', 'B1', 'B1', 'B1', 'A2']

ASSIGNMENTS:

First version:  [2, 3, 5, 6, 5, 2, 3, 3, 4, 5, 6, 6, 6, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2,
1, 3, 4, 3, 3, 3, 1, 2, 1, 2, 2, 2, 2, 2, 6, 6, 6, 1, 2, 3, 3, 3, 4, 1, 3, 5, 5, 2, 2, 3,
3, 3, 3, 3, 3, 5, 5, 6, 5, 5, 6, 6, 6, 6, 6, 6, 5, 2, 4, 4, 3, 3, 3, 4, 4, 4, 3, 3, 3]

Second version: [2, 1, 4, 5, 4, 1, 2, 2, 3, 4, 4, 4, 5, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2,
1, 1, 2, 2, 2, 3, 1, 3, 1, 2, 1, 1, 1, 2, 4, 4, 3, 2, 3, 4, 4, 4, 3, 2, 5, 5, 5, 2, 2, 2,
2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 4, 5, 3, 4, 4, 2, 3, 2, 3, 3, 3, 3, 3, 2]
```

Figura 5.1: Asignación de valores a niveles.

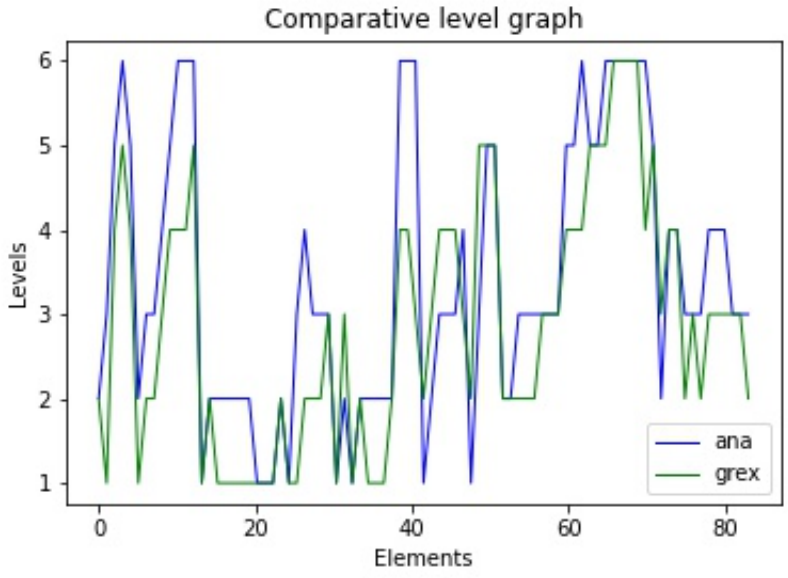

Figura 5.2: Gráfica comparativa de niveles de versiones.

```
First version: [2, 3, 2, 3, 3, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 3,
3, 3, 3, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 3, 3, 3, 1, 3, 2, 2, 3, 3, 3,
3, 3, 3, 2, 3, 3, 3, 3, 3, 3]

Second version: [2, 1, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 1,
1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1,
1, 1, 1, 2, 1, 1, 1, 1, 1, 1]
```

Figura 5.3: Valores de elementos considerados hasta el nivel B1.

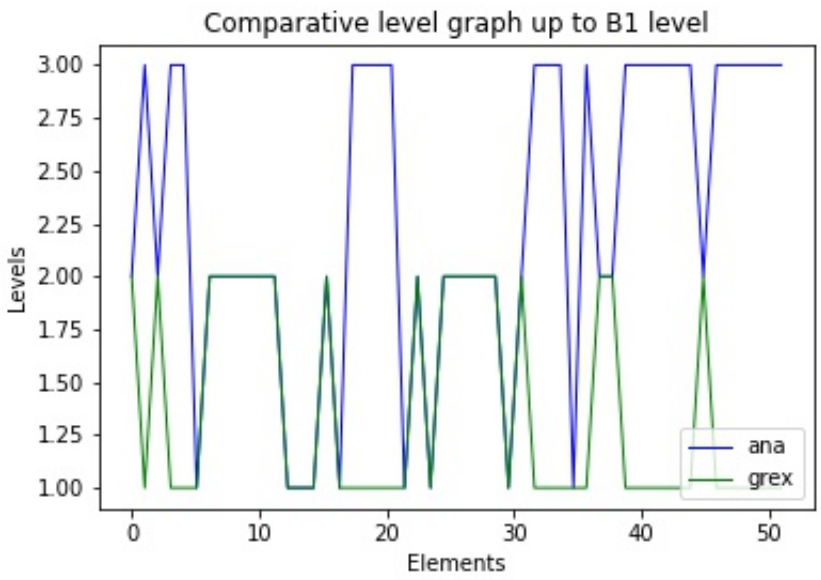

Figura 5.4: Gráfica comparativa de niveles hasta un nivel B1 de versiones.

TFG un nivel *B1-B2* en Python, es más difícil intuir los niveles *C1-C2* por lo que hay menor concordancia en niveles mayores. Por ello, se decide hacer un segundo cálculo del coeficiente de kappa, pero esta vez solo comparando los niveles hasta el B1 establecido en la primera versión como podemos ver en la figura 5.3, y se obtiene un resultado de *0.399* y una gráfica que la podemos observar en la figura 5.4. El resultado ha mejorado, y haciendo referencia a la tabla de la figura 3.3 se podría considerar como de *concordancia justa*.

Además, la diferencia de clases y niveles entre versiones más detallada se puede observar en las secciones B.1 y B.2 en el anexo B.

5.4. Validación de niveles

5.4.1. Creación de una encuesta

Para contrastar las ideas y obtener una primera opinión de *pythonistas*[1], se decide crear una encuesta mediante un formulario de *Google Forms* que servirá para realizar una encuesta.

Esta encuesta está comienza con unas preguntas de autoevaluación, como saber el nivel de Python y otros lenguajes, siendo los niveles disponibles los siguientes:

- A – Basic user
  - A1 – Breakthrough or beginner
  - A2 – Waystage or elementary
- B – Independent user
  - B1 – Threshold or intermediate
  - B2 – Vantage or upper intermediate
- C – Proficient user
  - C1 – Effective operational proficiency or advanced
  - C2 – Mastery or proficiency

Luego, sigue con un cuestionario para asignar el nivel considerado a los diferentes elementos extraídos del lenguaje de Python vistos en la sección 5.2.

Unas primeras versiones de la encuesta se enviaron a varios profesores y estudiantes de la ETSIT-URJC que sabemos que son *pythonistas*. Con sus opiniones se decide crear dos formatos de encuesta, una versión corta[2] y una versión larga[3], dependiendo del número de elementos de Python a evaluar. La razón de tener dos encuestas es que la larga lleva alrededor de 15 minutos, algo que algunos consideraron como excesivo, especialmente si queríamos obtener muchas respuestas. Así, la encuesta corta está diseñada para poder rellenarse en 5 minutos.

[1] A los desarrolladores versados en Python se los conoce popularmente como *pythonistas*

[2] https://forms.gle/Lun67oV2KmATsSMS6

[3] https://forms.gle/Fv7drg87sb5WHqV69

5.4.2. Resultados de la encuesta

Uno de los objetivos específicos de nuestro proyecto, era verificar el uso de CEFRL con 'pythonistas' mediante encuestas. Para ello se crean dos encuestas:

- Versión larga. Tiene todos los elementos a evaluar.
- Versión corta. Tiene los elementos más significativos.

Estas encuestas se lanzan por las redes sociales y se obtienen diferentes resultados. Entre los resultados obtenemos que:

1. El uso más predominante de Python es en el *trabajo/universidad*
2. El nivel medio de los usuarios es *B2 - Vantage or upper intermediate*
3. Los usuarios saben otros lenguajes a parte de Python como pueden ser Java, JavaScript, C#, C++, etc.
4. Se introducen elementos de 'control' como lo son *'Variadic Generics'* y *'Syntactic Macros'*, y de media eligen la opción de *Don't know*, que sería la opción correcta. Se pueden observar las gráficas de ambos elementos en las figuras 5.5 y 5.6, en ellas se puede apreciar que hay variedad de respuestas pero la predominante es *Don't know*. Además, *Variadic Generics* obtiene mayor número de respuestas debido a que este elemento estaba en ambas encuestas, a diferencia del otro elemento de control, que se introdujo solo en la encuesta larga.
5. En general, los usuarios no conocen las *metaclases*.

Además, se comprueba que es difícil establecer unos niveles a los elementos de forma consolidada, esto es debido a que cada usuario según su nivel y el uso que le da a este lenguaje, tiene diferente criterios de evaluación a diferencia de otro usuario. Cabe señalar también, como ya vimos en la sección anterior, que cuando un usuario no conoce un elemento tiende a establecer más nivel de lo que probablemente le corresponde. Por ello, se realizaron unas gráficas para mostrar los elementos correspondientes a cada nivel con su respectivo número de usuarios que establecieron ese valor para ese elemento. Estas gráficas se pueden observar en las figuras 5.7, 5.8, 5.9, 5.10, , 5.11 y 5.12.

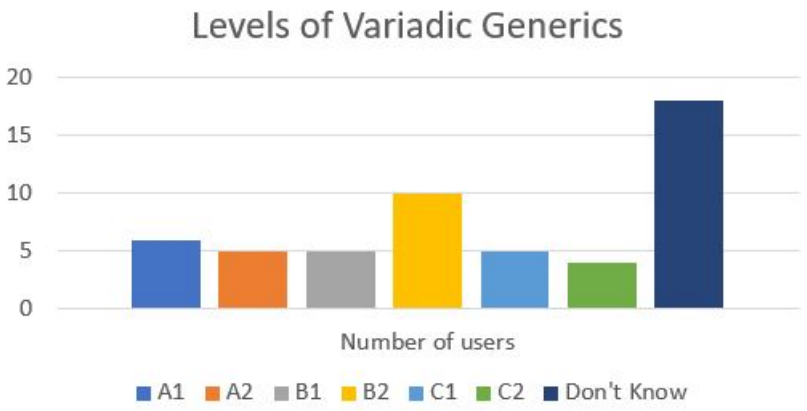

Figura 5.5: Gráfica de niveles de *Variadic Generics*.

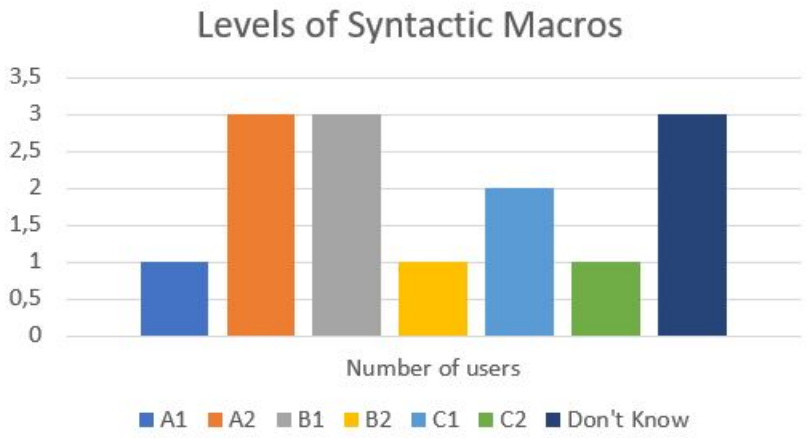

Figura 5.6: Gráfica de niveles de *Syntactic Macros*.

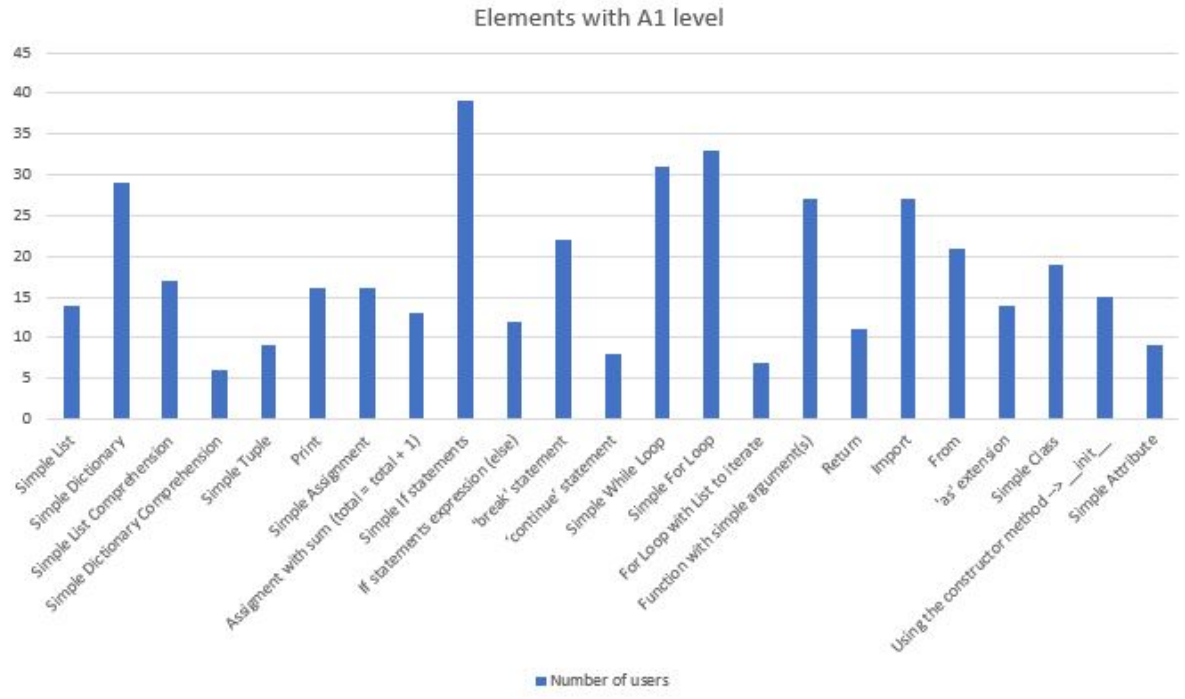

Figura 5.7: Gráfica de número de usuarios para cada elemento del nivel A1.

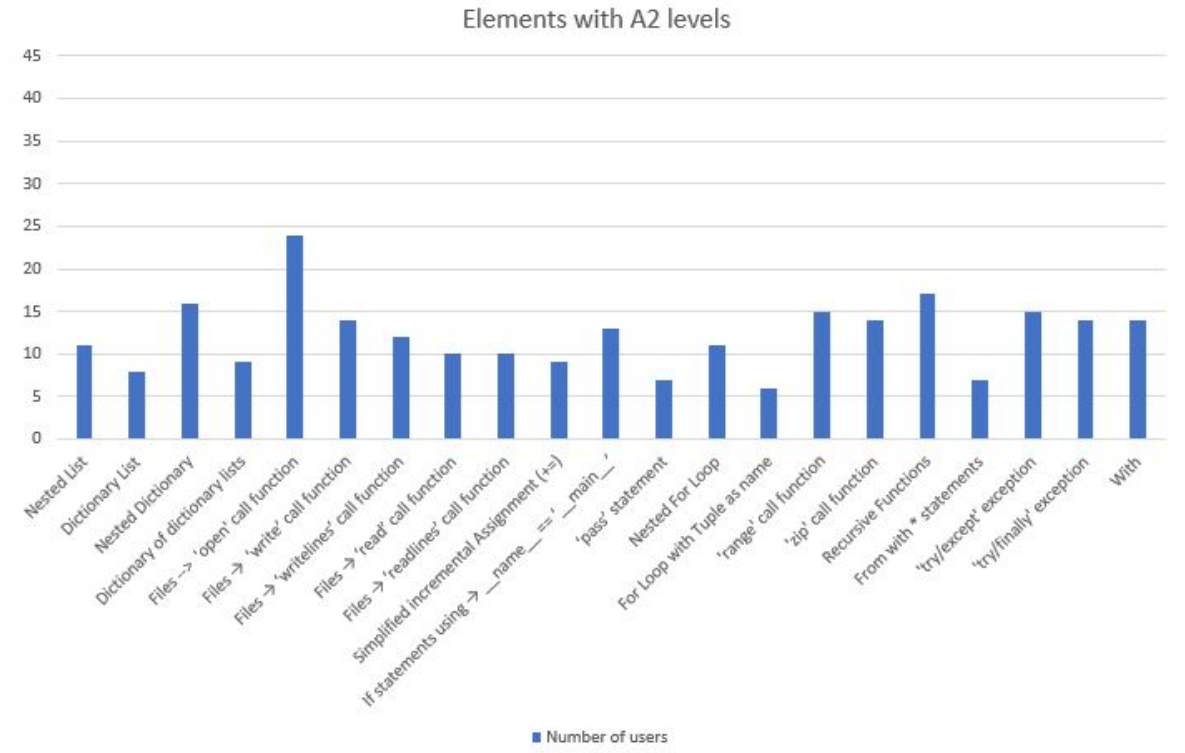

Figura 5.8: Gráfica de número de usuarios para cada elemento del nivel A2.

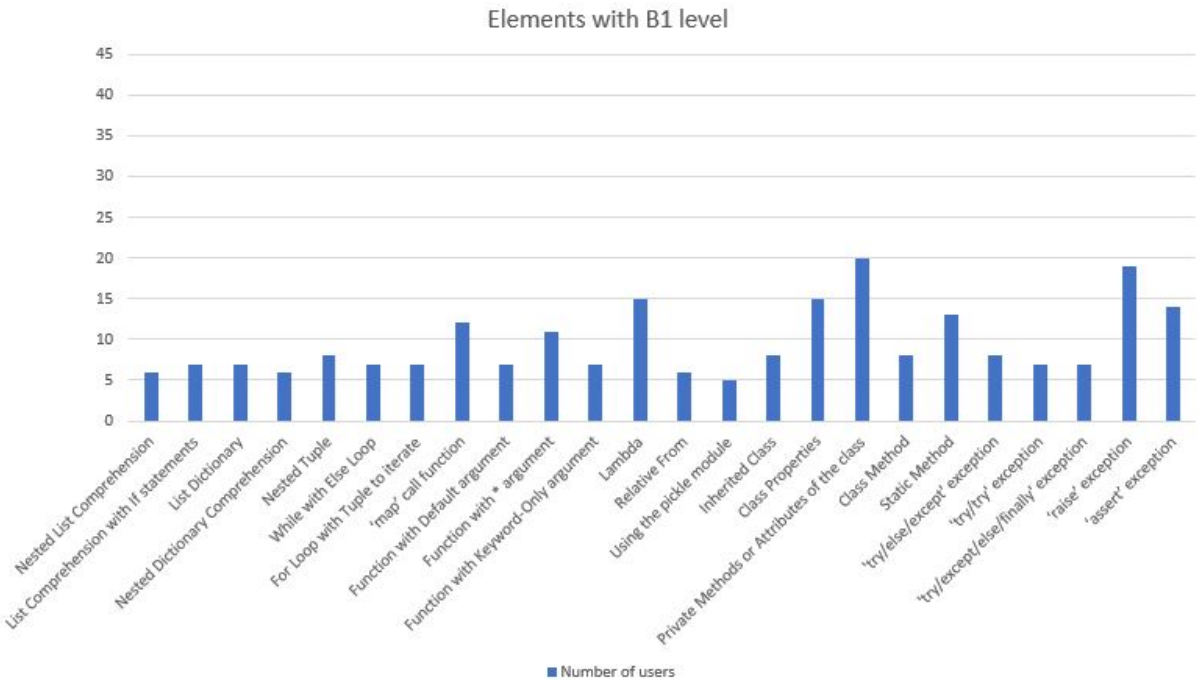

Figura 5.9: Gráfica de número de usuarios para cada elemento del nivel B1.

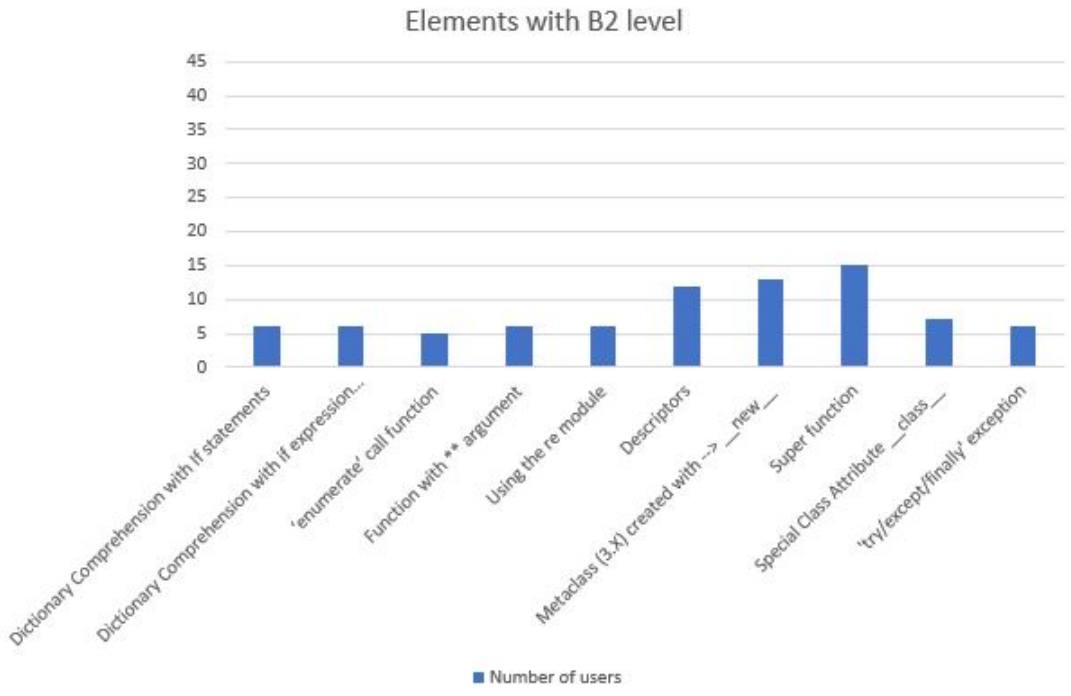

Figura 5.10: Gráfica de número de usuarios para cada elemento del nivel B2.

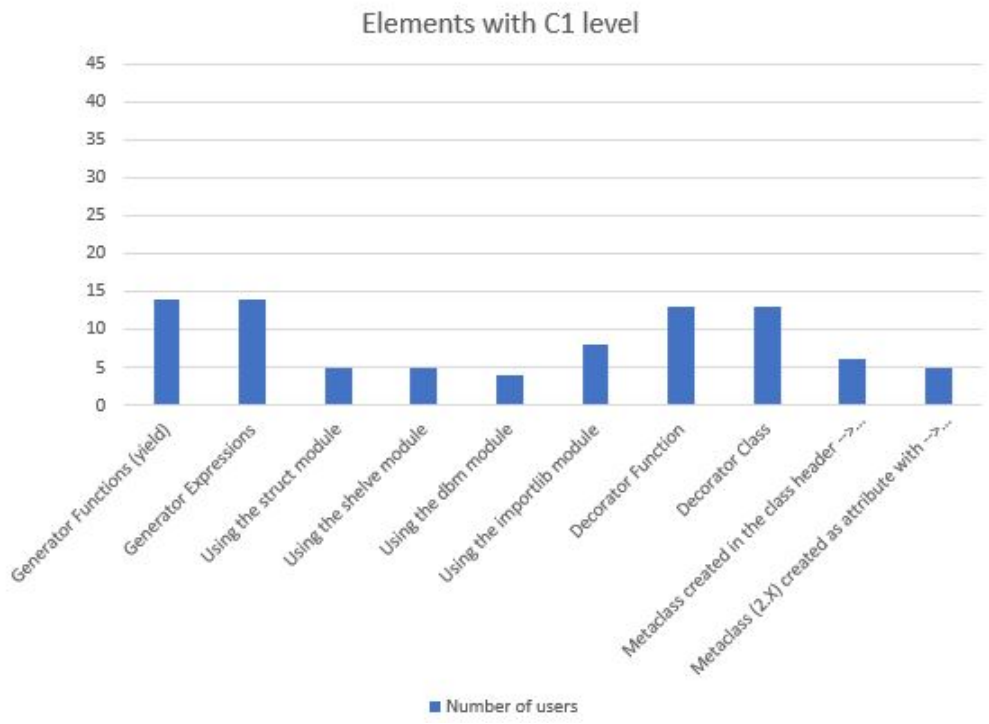

Figura 5.11: Gráfica de número de usuarios para cada elemento del nivel C1.

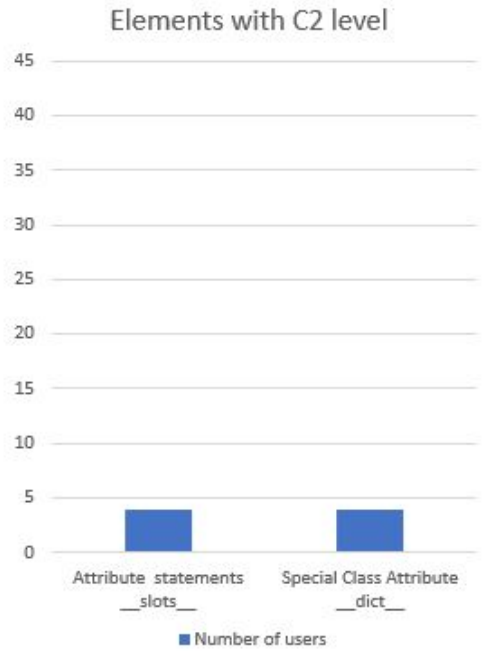

Figura 5.12: Gráfica de número de usuarios para cada elemento del nivel C2.

En ellas, se puede comprobar que hay elementos que obtienen mayor número de votaciones, esto es debido a que había elementos comunes en ambas encuestas, obteniendo así estos mayor número de usuarios. Además, se puede observar que algunos elementos como pueden ser *Simple List Comprehension* y *Simple Class* en el caso del nivel A1, no sería un nivel correcto para estos elementos ya que deberían haber obtenido un nivel más elevado. Esto ocurre también con las *metaclasses*, que están en un nivel *B2-C1* cuando sería más correcto un *C1-C2*. Tras esto, se tuvo que hacer un filtrado de respuestas para poder obtener unos niveles más correctos asignados a cada elemento.

5.4.3. Cálculo de coeficiente de Kappa

Como ya vimos en la sección 5.3, se hizo un cálculo del *coeficiente de kappa* entre dos versiones propuestas. Con las encuestas tenemos más de dos observadores, por lo que se calculará un nuevo coeficiente de kappa pero esta vez de *Fleiss*, que está diseñado para estas situaciones. Para ello, se va a hacer ese cálculo para cada tipo de encuesta ya que la versión corta tiene menos elementos.

Antes de nada se hace un filtrado de las respuestas ya que había respuestas *no válidas* como pueden ser:

- Asignar el nivel más alto, *C2*, a una estructura básica como puede ser una *Simple List*.
- Asignar un mismo nivel a todo, debido a que había sido contestada la encuesta aleatoriamente o sin entender bien su objetivo.
- Elegir la opción de *'Don't know'* en todos los elementos.

A continuación, se realiza el cálculo de dos maneras distintas:

1. **Opción 1**: Usando la opción *'Don't know'* como un valor más.
2. **Opción 2**: Asignando a la opción *'Don't know'* el valor del usuario anterior.

Los resultados se pueden observar en la figura 5.13, en la cual se puede comprobar que los resultados se encuentran en la franja de *concordancia ligera*. Los resultados mejoran en ambas encuestas, cuando aplicamos la *opción 1* ya que de esta manera provoca mayor similitud entre

Survey	Option 1	Option 2
Long	0,151	0,145
Short	0,186	0,177

Figura 5.13: Resultados del cálculo del coeficiente de kappa de Fleiss para seis niveles.

Survey	Option 1	Option 2
Long	0,25	0,235
Short	0,182	0,166

Figura 5.14: Resultados del cálculo del coeficiente de kappa de Fleiss para tres niveles.

usuarios. También destacar, que la encuesta corta obtiene mejor resultado debido a que posee menos elementos que comparar.

Tras esto, al ver que los resultados no eran buenos, se agruparon los subniveles en niveles. Es decir, *A1-A2* en *A*, *B1-B2* en *B* y *C1-C2* en *C*, de esta manera, evitar la diferencia entre subniveles pertenecientes al mismo nivel y poder obtener así mayor concordancia. Los resultados obtenidos se pueden observar en la figura 5.14, comprobando así que en la encuesta corta no varía apenas el resultado, incluso empeora, pero en la larga si se puede apreciar, pasando de concordancia ligera a justa como se evalúa en la figura 3.3 debido a que aumenta la similitud entre usuarios.

5.4.4. Elaboración de la configuración *'vanilla'*

Tras realizar numerosos cálculos y analizar las diferentes respuestas, se decide elaborar una versión final de niveles, la que llamaremos configuración *'vanilla'* o configuración por defecto. Para ello, se recogen las respuestas más significativas de la encuesta larga y se decide calcular la media de niveles y su desviación típica.

De esta manera, al calcular la media de niveles para cada elemento, sacamos el nivel final para cada uno de ellos, algunos de ellos son los siguientes:

- **Elementos con nivel A1:**

1. Simple List
2. Simple Dictionary
3. Return

- **Elementos con nivel A2:**

1. Simple Tuple
2. Files: 'open' call function
3. 'continue' statement: A2

- **Elementos con nivel B1:**

1. Nested Dictionary
2. Function with Keyword-Only argument
3. Simple Class

- **Elementos con nivel B2:**

1. Simple List Comprehension
2. Lambda
3. 'try/finally' exception

- **Elementos con nivel C1:**

1. Generator Functions (yield)
2. Descriptors
3. Decorator Function

- **Elementos con nivel C2:**

1. Metaclass (3.X) created with __new__
2. Attribute statements __slots__
3. Using the struct module

Además, uno de los objetivos principales, era obtener un *ranking* de los elementos más problemáticos al evaluar, es decir, que tuvieran mayor grado de desacuerdo, por eso se realiza la desviación típica de cada elemento comparando las respuestas de cada usuario.

En Python, la desviación típica se calcula con el módulo *'statistics'*[4] haciendo uso del método *'stdev'*. Gracias a esto, se obtiene que los elementos con mayor acuerdo son *Simple Assignment* y *Simple If statements* con un resultado de *0.0*; y los más problemáticos son *Metaclass (2.X) created as attribute with __metaclass__* y *Using the pickle module* con unos resultados de *2.3* y *2.6* respectivamente.

Esta versión de niveles, se encuentra en el anexo A en la sección A.4 en las figuras A.10, A.11 y A.12.

A continuación, como ya se vio en el apartado anterior, los cálculos del coeficiente de kappa de Fleiss no habían sido buenos, por lo que se decide calcular de nuevo el valor de kappa, pero esta vez de *Cohen* como se realizó en la sección 5.3. De esta manera, se obtiene la concordancia entre la configuración *vanilla* y cada usuario considerado en la elaboración de esta sin considerar los elementos que tenían como respuesta *'Don't know'*.

Algunos de los resultados que se obtienen para la comparación de seis niveles (*A1, A2, B1, B2, C1* y *C2*), se pueden observar en la figura 5.15, comprobando que de media se obtiene un valor de *0.4* siendo así una *concordancia moderada*, mejorando así lo obtenido hasta ahora; y para tres niveles (*A, B* y *C*) que se muestran en la figura 5.16, donde mayoritariamente, excepto casos aislados, al realizar la medición con menos niveles los resultados mejoran hasta un *0.6* aproximadamente, siendo así una *concordancia sustancial*.

Como se comprueba en los resultados anteriores, hay usuarios que no obtienen buena concordancia, por ello, se decide eliminarlos tanto de la elaboración de la configuración *vanilla* como de los cálculos de kappa. Tras esto, se elabora una segunda versión de esta configuración, donde aparecen nuevos elementos con una desviación típica de *0.0* como son los elementos *Print*, *From* y *'raise' exception*; y los más problemáticos varían, siendo el peor *Using the pickle module* esta vez con un resultado de *2.4* mejorando ligeramente, y el valor anterior de *Metaclass (2.X) created as attribute with __metaclass__* mejora hasta un valor de *0.54* a diferencia del anterior caso que era *2.3*. Además, algunos elementos varían en el nivel, como los siguientes:

- Nested List: *B1* $\rightarrow$ *A2*.

[4] https://docs.python.org/3/library/statistics.html

```
Kappa coefficient of user 1: 0.4874055415617128
Kappa coefficient of user 2: 0.4696272799365582
Kappa coefficient of user 3: 0.4158590308370044
Kappa coefficient of user 4: 0.4496472097498396
Kappa coefficient of user 5: 0.1815266196279668
Kappa coefficient of user 6: 0.26865671641791034
Kappa coefficient of user 7: 0.5433962264150942
```

Figura 5.15: Resultados del cálculo del coeficiente de kappa de Cohen para seis niveles (I).

```
Kappa coefficient of user 1: 0.6226856363282012
Kappa coefficient of user 2: 0.638157894736842
Kappa coefficient of user 3: 0.6162528216704288
Kappa coefficient of user 4: 0.586009408877071
Kappa coefficient of user 5: 0.09427748044462736
Kappa coefficient of user 6: 0.37244897959183676
Kappa coefficient of user 7: 0.6492626544440017
```

Figura 5.16: Resultados del cálculo del coeficiente de kappa de Cohen para tres niveles (I).

```
Kappa coefficient of user 1: 0.5382526029877772
Kappa coefficient of user 2: 0.5650860767139837
Kappa coefficient of user 3: 0.42393436775459237
Kappa coefficient of user 4: 0.47857905168470805
Kappa coefficient of user 5: 0.4201378483667966
```

Figura 5.17: Resultados del cálculo del coeficiente de kappa de Cohen para seis niveles (II).

- List Comprehension with If statements: *B2* → *C1*.
- 'break' statements: *A1* → *A2*.
- Function with simple argument(s): *A1* → *A2*.
- Lambda: *B2* → *C1*.
- 'as' extension: *B1* → *A2*.
- Using the struct module: *C2* → *C1*.
- Using the pickle module y Using the pickle module: *'Don't know'* → *C2*.
- Attribute statements __slots__: *C2* → *C1*.
- Special Class Attribute __dict__: *C2* → *B2*.
- 'try/finally' exception: *B2* → *B1*.

Por último, se obtienen los nuevos valores del cálculo del coeficiente de kappa, para seis niveles que se muestran en la figura 5.17 obteniendo de media un valor de *0.5* aproximadamente, siendo una *concordancia moderada* y habiendo mejorado un *0.1* con respecto al caso anterior; y para tres niveles que se pueden observar en la figura **??**, donde los resultados mejoran hasta un *0.66* aproximadamente, con una *concordancia sustancial* y variando ligeramente con respecto al anterior análisis.

”

””””

Capítulo 6

Resultados y experimentos

6.1. Pruebas con usuarios de GitHub

El objetivo de este proyecto es obtener un análisis de programas en Python para obtener un resumen de niveles. Para ello, se ha realizado la prueba con los repositorios de la asignatura de PTAVI (*"Protocolos para la Transmisión de Audio y Vídeo por Internet"*) del primer cuatrimestre de tercero del Grado en Ingeniería de Sistemas Audiovisuales y Multimedia, en concreto con los repositorios de la autora de este TFG. En esta asignatura se programa en Python, por lo que se decide evaluar las diferentes prácticas para obtener un análisis sobre ellas.

Lo primero que se hace es introducir los parámetros en nuestro programa principal:

```
python3 main.py user anapgh
```

Tras esto, nuestro programa obtendrá todos los repositorios que se encuentran en el usuario y analizará en ellos si se obtiene el 50 % de lenguaje de Python como podemos observar en la figura 6.1. Y efectivamente, solo se clonan los repositorios que superan ese porcentaje como se puede observar en la figura 6.2. Después de clonar, se muestra la ubicación donde se ha descargado este repositorio, muestra los archivos disponibles en él y hace un segundo filtro, tras el cual solo analiza los archivos que terminan en *.py*.

Después de realizar el análisis de todos los repositorios de la asignatura que se muestran en la figura 6.3, se obtiene el resultado del análisis por la *shell* como se observa en la figura 6.4 y los ficheros de resultados en formato JSON y CSV.

```
(base) ana@ana-Lenovo-Flex-3-1470:~/Documentos/TFG/TFG$ python3 main.py user anapgh
https://api.github.com/users/anapgh
Analyzing user...

Analyzing repositories...


Repository: 2019-2020-CSAAI-Practicas
Analyzing repository languages...

JavaScript: 119072
HTML: 23425
CSS: 9899

Repository: 2019-2020-GyV3D
Analyzing repository languages...

HTML: 93492
JavaScript: 10584
CSS: 389

Repository: 2020-2021-LTAW-Practicas
Analyzing repository languages...

JavaScript: 101678
HTML: 37066
CSS: 14528

Repository: Fuentes
Analyzing repository languages...


Repository: ptavi-final
Analyzing repository languages...

Python: 28769

Python 50% OK
```

Figura 6.1: Comprobando lenguajes de programación.

```
Repository: ptavi-final
Analyzing repository languages...

Python: 28769

Python 50% OK

Run url...
Clonando en 'ptavi-final'...
remote: Enumerating objects: 218, done.
remote: Counting objects: 100% (218/218), done.
remote: Compressing objects: 100% (89/89), done.
remote: Total 218 (delta 126), reused 218 (delta 126), pack-reused 0
Recibiendo objetos: 100% (218/218), 321.00 KiB | 1011.00 KiB/s, listo.
Resolviendo deltas: 100% (126/126), listo.
The directory is: ptavi-final
This script absolute path is  /home/ana/Documentos/TFG/TFG/ptavi-final
Directory:
['uaclient.py', 'pr.xml', 'uaserver.py', '.git', 'ua2.xml', '.gitignore', 'avanzadas.
txt', 'ua1.xml', 'error.libpcap', 'mp32rtp', 'README.md', 'proxy_registrar.py', 'llam
ada.libpcap', 'cancion.mp3', 'passwords.json', 'LICENSE', 'check-pfinal.py']
Python File: uaclient.py
Python File: uaserver.py
Python File: proxy_registrar.py
Python File: check-pfinal.py
```

Figura 6.2: Clonando repositorios.

Figura 6.3: Repositorios clonados.

```
====================================
RESULT OF THE ANALYSIS:
Analyzed .py files: 22
Elements of level A1: 802
Elements of level A2: 676
Elements of level B1: 76
Elements of level B2: 14
====================================
```

Figura 6.4: Resultado del análisis por la shell.

	A	B	C	D	E	F	G
1	Repository	File Name	Class	Start Line	End Line	Displacement	Level
2	ptavi-final	uaclient.py	Simple List	22	22	19	A1
3	ptavi-final	uaclient.py	Simple List	119	119	35	A1
4	ptavi-final	uaclient.py	Simple List	120	120	36	A1
5	ptavi-final	uaclient.py	Simple List	121	121	36	A1
6	ptavi-final	uaclient.py	Simple List	122	122	36	A1
7	ptavi-final	uaclient.py	Simple List	123	123	31	A1
8	ptavi-final	uaclient.py	Simple List	124	124	33	A1
9	ptavi-final	uaclient.py	Simple Tuple	165	165	11	A1
10	ptavi-final	uaclient.py	Simple Tuple	47	47	15	A1
11	ptavi-final	uaclient.py	Simple Tuple	221	221	30	A1
12	ptavi-final	uaclient.py	Simple Dictionary	117	117	20	A2
13	ptavi-final	uaclient.py	6 List Dictionary	118	125	24	B1
14	ptavi-final	uaclient.py	Print	140	140	4	A1
15	ptavi-final	uaclient.py	Print	264	264	4	A1
16	ptavi-final	uaclient.py	Files --> 'open' call function	175	175	17	A2
17	ptavi-final	uaclient.py	'range' call function	23	23	21	A2
18	ptavi-final	uaclient.py	Files --> 'open' call function	63	63	13	A2
19	ptavi-final	uaclient.py	Files --> 'write' call function	64	64	12	A2
20	ptavi-final	uaclient.py	Print	225	225	12	A1
21	ptavi-final	uaclient.py	Simple Atributte	18	18	8	A2
22	ptavi-final	uaclient.py	Simple Atributte	24	24	8	A2
23	ptavi-final	uaclient.py	Simple Atributte	50	50	15	A2
24	ptavi-final	uaclient.py	Simple Atributte	59	59	8	A2
25	ptavi-final	uaclient.py	Simple Atributte	117	117	8	A2
26	ptavi-final	uaclient.py	Simple Atributte	118	118	8	A2

Figura 6.5: Resultado de análisis en formato CSV.

En el fichero CSV de la figura 6.5, se puede observar las distintas columnas con los valores de *'Repository', 'File Name', 'Class' , 'Start Line', 'End Line', 'Displacement'* y *'Level'* como se muestra en la figura 6.5. Además, se va obtener un fichero.csv por cada fichero analizado que se almacenará en la carpeta en la carpeta *'DATA_CSV'* como se muestra en la figura 6.6.

Del formato JSON obtenido mostrado en la figura 6.7, se va obtener derivados de él, como vimos en la sección 4.5. Estos archivos.json se van a encontrar en la carpeta *'DATA_JSON'* como se muestra en la figura 6.8.

Los archivos en formato JSON creados se utilizan para mostrar los resultados de forma más visual en forma de página web como ya se explicó con anterioridad. Para ello, se cuenta con los cuatro archivos mencionados en la sección 4.6, y se ejecuta en la *shell* el siguiente comando:

```
node main.js
```

Y automáticamente se crean los archivos deseados, *'index.html'* y los correspondientes a cada repositorio, como se puede ver en la figura 6.9. Con estos archivos creados, al lanzar en el navegador el archivo HTML *index*, se visualiza una página con lo siguiente:

- Un botón por cada repositorio analizado.
- Un resumen general de los niveles y clases de todos los repositorios.

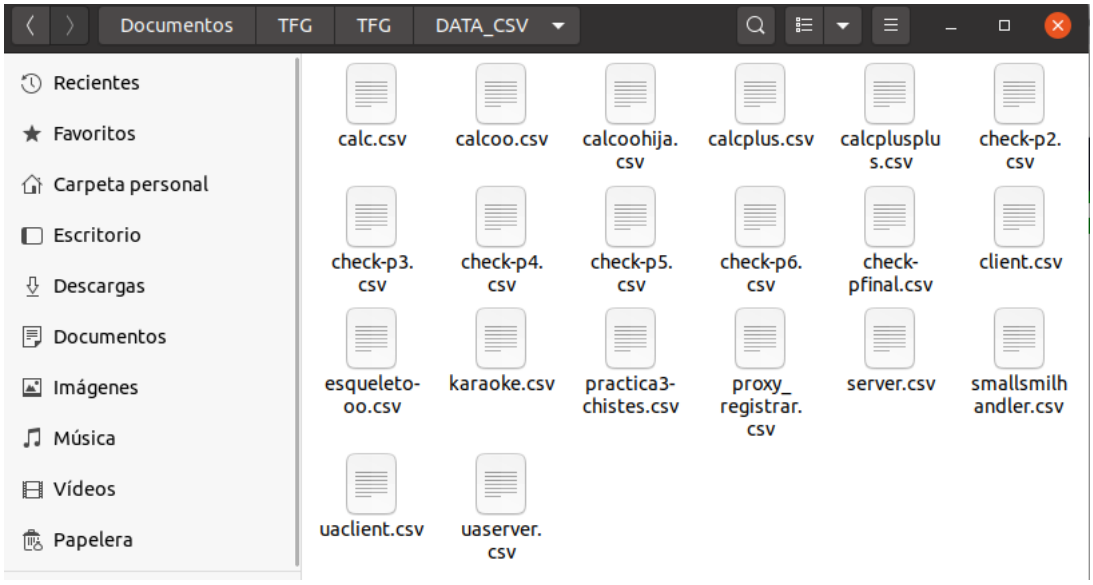

Figura 6.6: Archivos csv.


```
{
    "ptavi-final": {
        "uaclient.py": [
            {
                "Class": "Simple List",
                "Start Line": "22",
                "End Line": "22",
                "Displacement": "19",
                "Level": "A1"
            },
            {
                "Class": "Simple List",
                "Start Line": "119",
                "End Line": "119",
                "Displacement": "35",
                "Level": "A1"
            },
            {
                "Class": "Simple List",
                "Start Line": "120",
                "End Line": "120",
                "Displacement": "36",
                "Level": "A1"
            },
            {
                "Class": "Simple List",
                "Start Line": "121",
                "End Line": "121",
                "Displacement": "36",
                "Level": "A1"
            },
            {
                "Class": "Simple List",
                "Start Line": "122",
                "End Line": "122",
                "Displacement": "36",
                "Level": "A1"
```


Figura 6.7: Resultado de análisis en formato JSON.

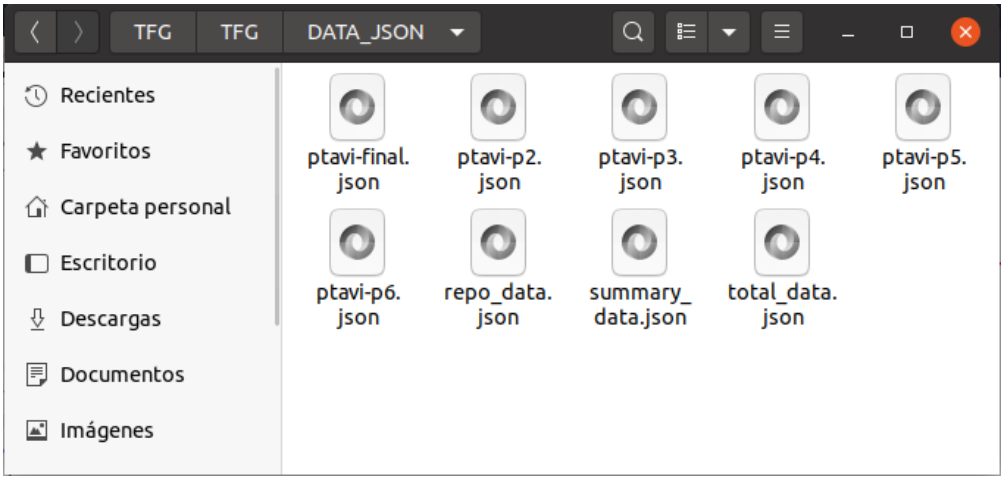

Figura 6.8: Archivos json.

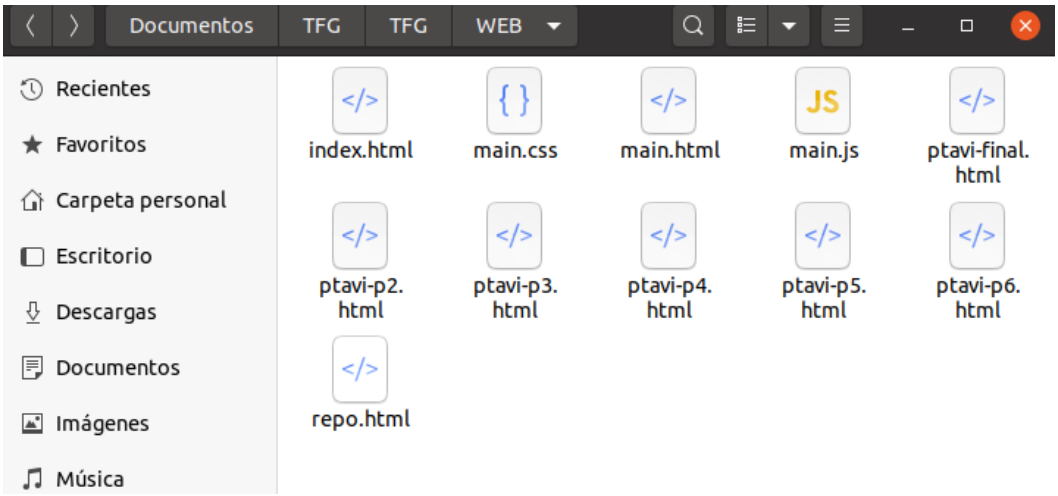

Figura 6.9: Archivos de la página web.

Esto lo podemos comprobar en la figura 6.10. Además, si hacemos *click* en algunos de los botones, nos redirige al repositorio correspondiente, el cual nos mostrará un resumen general como se puede ver en la figura 6.11 y un análisis detallado del repositorio mostrado en la figura 6.12.

6.2. Resultados del análisis a nivel usuario

Tras realizar el análisis del usuario *anapgh* de los repositorios de la asignatura de PTAVI, se ha comprobado lo siguiente:

- No hay una gran variación de niveles entre el repositorio de la práctica primera, *ptavi-p2* y

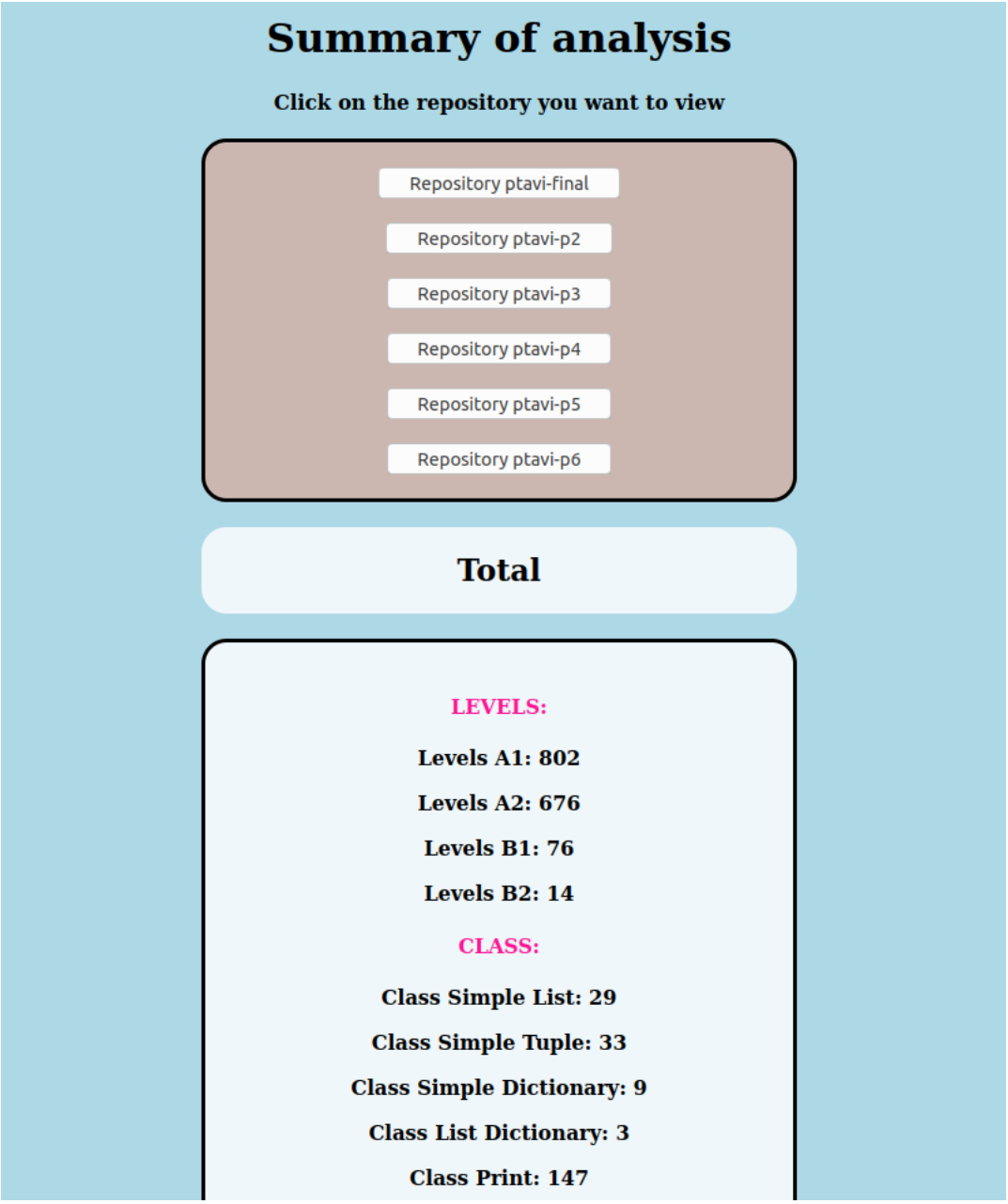

Figura 6.10: Visualización de la página web principal.

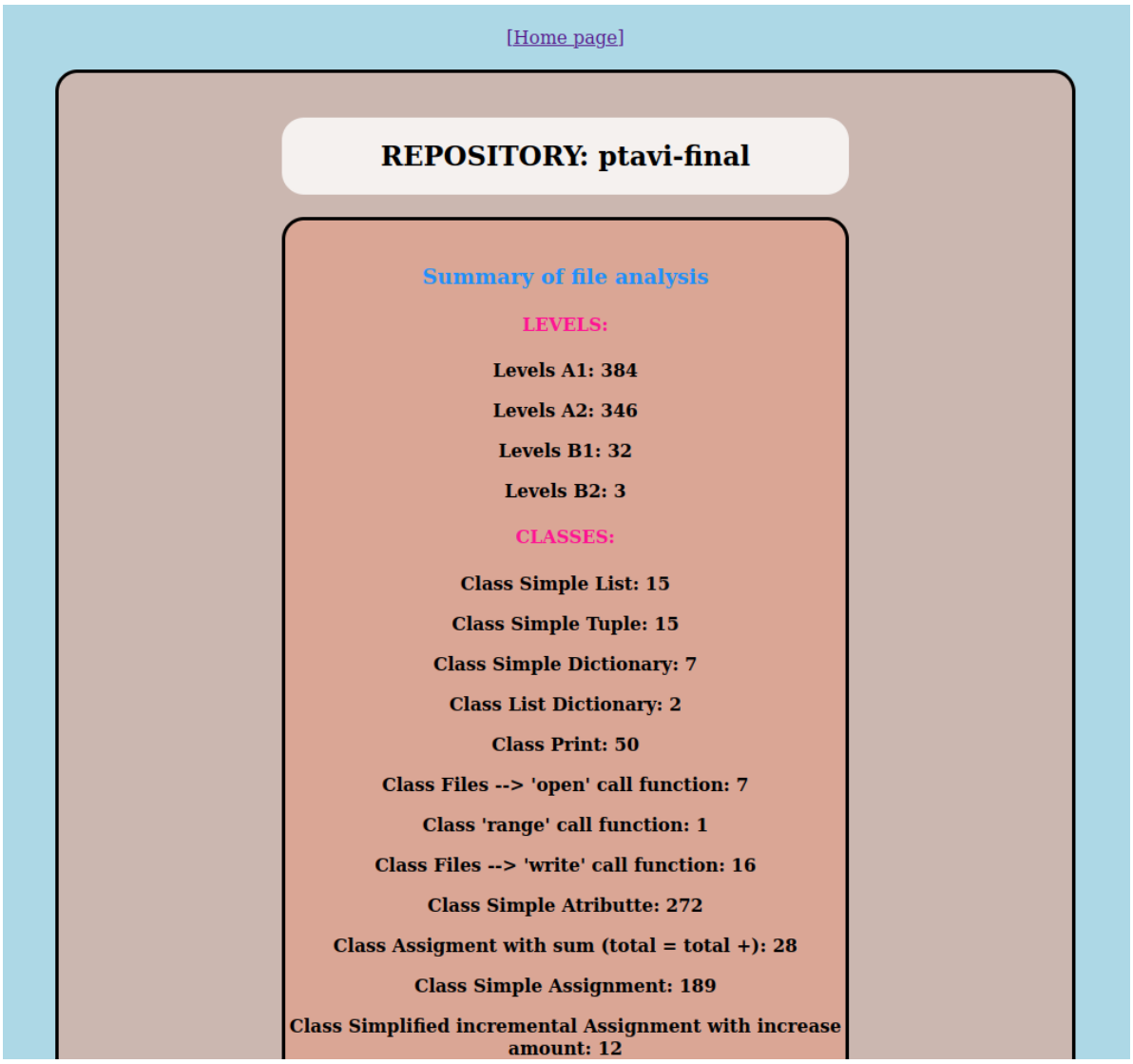

Figura 6.11: Visualización del resumen de la página del repositorio.

Figura 6.12: Visualización del análisis detallado del repositorio.

LEVELS			
ptavi-p2		ptavi-pfinal	
A1	117	A1	384
A2	85	A2	346
B1	15	B1	32
B2	6	B2	3

Figura 6.13: Comparación de niveles entre repositorios. a) ptavi-p2. b) ptavi-pfinal.

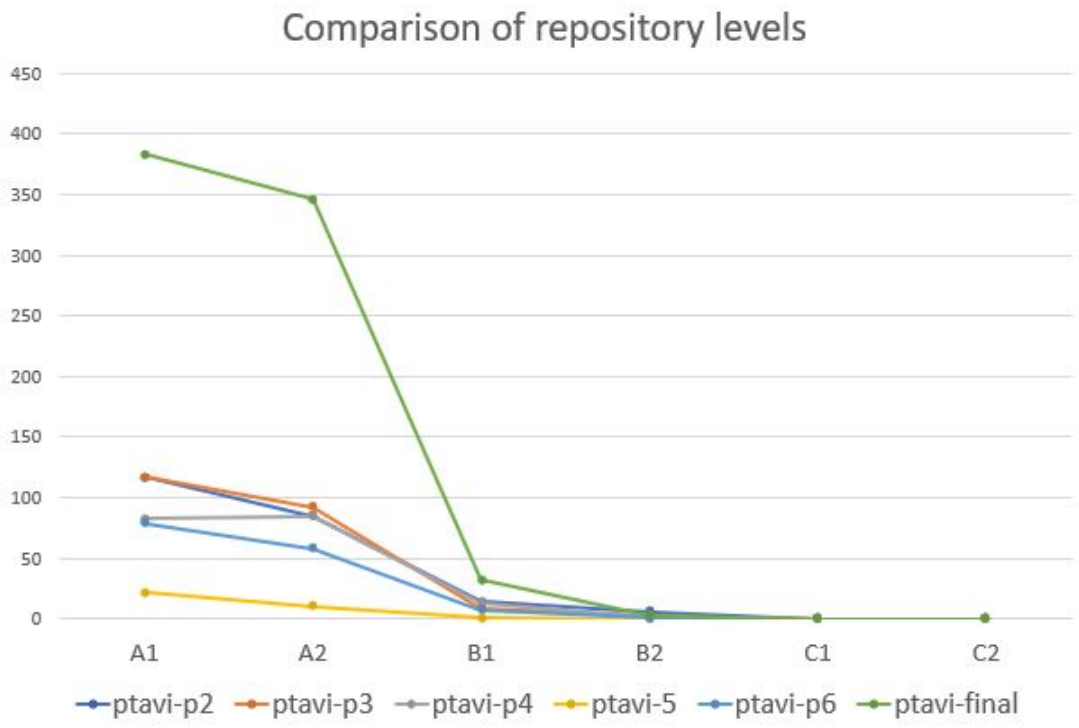

Figura 6.14: Comparación de niveles de repositorios analizados.

el repositorio del proyecto final de la asignatura *ptavi-pfinal*. Esto lo podemos observar en la figura 6.13. Esto es debido a que la asignatura se centra más en aprender los diferentes protocolos de redes, que en la complejidad de programación.

- El nivel que se adquiere en esta asignatura de Python es de media un nivel B1, es decir, un nivel medio. Por lo que este resultado sería adecuado para una primera asignatura de este lenguaje. Se puede observar en la figura 6.14.

- Se comprueba que en los ficheros *'check.py'* creados y usados por el profesor para la correcta entrega, no aparece código con nivel C1, ya que se eliminaron elementos como

List Comprehension para no confundir al alumno.

- No hay mucha variación en las clases usadas en cada repositorio, esto se puede ver en la figura 6.15.

6.3. Otras pruebas con usuarios de GitHub de la encuesta

Se realizaron más pruebas con usuarios que nos dejaron su nombre de usuario en la encuesta de la sección 5.4. Esto se podía hacer de manera opcional y nos permite ver si su autoevaluación concuerda con lo que nos ofrece la herramienta. Obtuvimos los siguientes resultados:

- Usuario *1mikegrn*[1]. Se analiza su usuario entero, y su resultado se puede observar en la figura 6.16, obteniendo un análisis de 99 ficheros.py, los cuales contienen elementos de todos los niveles, desde A1 hasta C2. De esta manera, se comprueba que este usuario en la encuesta se autoevaluó bien, ya que estableció su nivel de Python con un nivel de *C2*.

- Usuario *citostyle*[2]. Se analiza su usuario entero, y su resultado se puede observar en la figura 6.17. Este usuario posee menos repositorios, por lo que se analizan solamente seis ficheros, en los cuales se puede apreciar que aunque sean menos, obtiene elementos de todos los niveles posibles. Se comprueba su nivel evaluado en la encuesta, y este se asignó un nivel *B2*, por lo que podría ser debido a que elementos de nivel *C1* y *C2* pudieron ser extraídos vía internet, o simplemente este usuario se autoevaluó por debajo de su nivel.

- Usuario *yebityon*[3]. Se analiza su usuario entero, y su resultado se puede observar en la figura 6.18. En este caso, solo analiza 4 ficheros, de los que se obtienen elementos de A1-A2 mayoritariamente. Luego aparecen elementos B1, y alguno C2, esto puede significar que este usuario tendrá un nivel B1 como máximo, y ha cogido elementos de mayor nivel extraídos de internet. Se compara ese nivel con el asignado en la autoevalución, y efectivamente este se asignó un nivel *B1*.

- Usuario *nnelluri928*[4]. Se analiza su usuario entero, y su resultado se puede observar en la

[1] https://github.com/1mikegrn
[2] https://github.com/citostyle
[3] https://github.com/yebityon
[4] https://github.com/nnelluri928

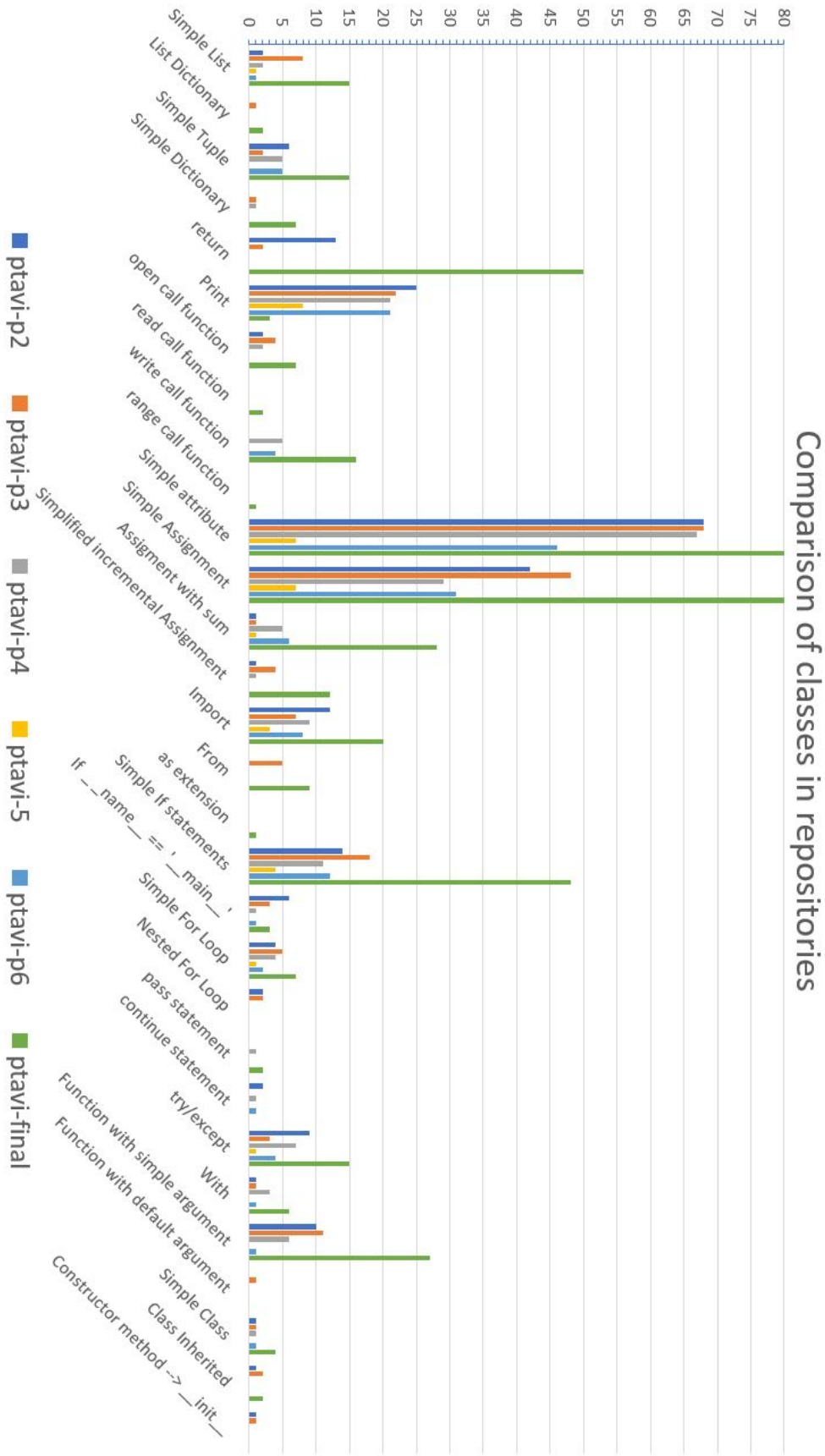

Figura 6.15: Comparación de clases de repositorios analizados.

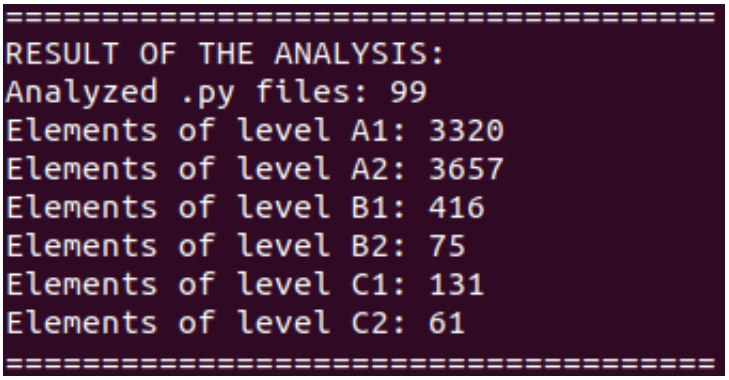

Figura 6.16: Análisis del usuario *1mikegrn*.

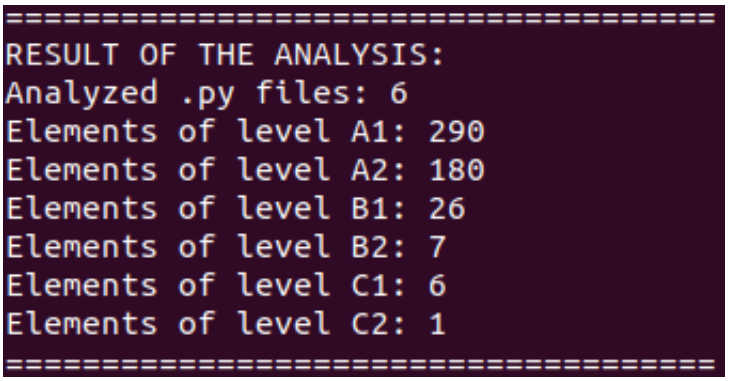

Figura 6.17: Análisis del usuario *citostyle*.

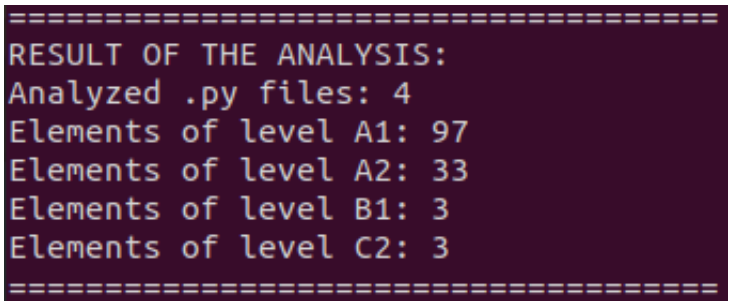

Figura 6.18: Análisis del usuario *yebityon*.

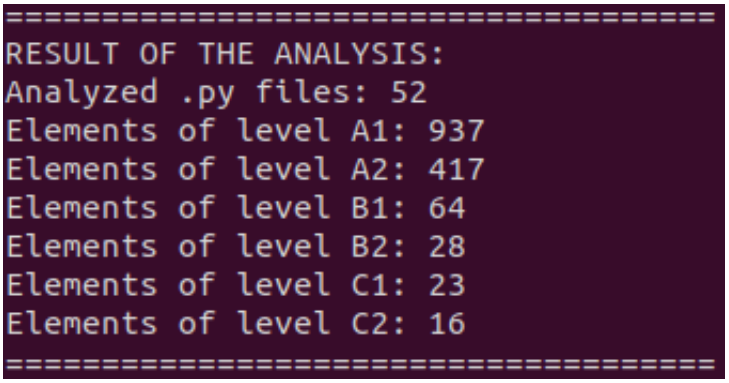

Figura 6.19: Análisis del usuario *nnelluri928*.

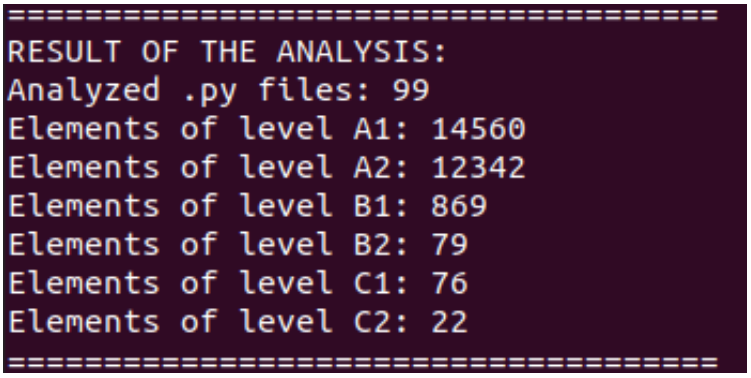

Figura 6.20: Análisis del usuario *jgbarah*.

figura 6.19, obteniendo así un análisis de 55 ficheros.py, que contienen todos los niveles correspondientes al CERFL, desde el A1 al C2. Se comprueba su autoevaluación, y es un nivel *B2*, por lo que esto puede ser debido a que posea más conocimientos de los que cree, se hayan analizado proyectos clonados con un *fork* o código extraído de internet.

- Usuario *jgbarah*[5]. Se analiza su usuario entero, y su resultado se puede observar en la figura 6.20. El análisis de este usuario requiere de mayor coste computacional debido a que tiene más repositorios, algunos de ellos muy extensos. Se obtiene el análisis de 99 ficheros.py, con todos los niveles correspondientes. Este usuario, se autoevaluó con un nivel *C1*, por lo que podría ser correcto hasta incluso tener este más nivel del asignado.

Con estos resultados, se pudo comprobar que estos usuarios tienen elementos en sus códigos de todos los niveles, desde el nivel *A1 (Breakthrough or beginner)* al *C2 (Mastery or proficiency)*. De esta manera, se pudo verificar la fiabilidad de nuestra herramienta extrayendo

[5]https://github.com/jgbarah

elementos de todos los niveles posibles.

Capítulo 7

Conclusiones

7.1. Consecución de objetivos

Haciendo referencia al capítulo 2, el objetivo principal de este TFG era implementar una herramienta capaz de obtener el nivel de Python de un usuario analizando sus códigos, basándose en los niveles del CEFRL. Este primer objetivo se ha cumplido, tal y como se esperaba.

Con respecto a los objetivos específicos, también se han cumplido. Empezando por hacer personalizada la elección de niveles para cada usuario que la utilice haciendo uso del fichero de configuración creado, la creación de una configuración *vanilla* de niveles de Python gracias a los resultados de los 'pythonistas' al rellenar la encuesta que se publicó. Además, con las pruebas a los alumnos de la asignatura de *ptavi* se obtuvo el nivel esperado para una primera asignatura de Python. Y por último, la creación de la de página web hizo posible una forma más amigable de poder visualizar los resultados del análisis.

Lo único que no se ha podido conseguir es analizar todos los códigos de Python disponibles, únicamente se ha podido analizar los asociados a Python3 debido a la limitación que nos ha proporcionado el módulo *ast* por su incompatibilidad con versiones anteriores al usado.

7.2. Aplicación de lo aprendido

Para la realización este proyecto, se ha usado los conocimientos de varias asignaturas cursadas en este grado como son las siguientes:

- **Informática I**. Primera asignatura de programación del grado, la cual me ayudó a saber

los elementos principales de un lenguaje y a estructurar un problema en subproblemas.

- **Informática II**. Una asignatura que aumento el nivel de programación proponiendo nuevos retos a cumplir.

- **Protocolos para la Transmisión de Audio y Vídeo por Internet *(ptavi)***. En esta asignatura aprendí el lenguaje de Python. Además, esta asignatura me motivó e hizo que cogiera el gusto a programar, por ello decidí que mi TFG debería ser orientado a este lenguaje.

- **Construcción de Servicios y Aplicaciones Audiovisuales en Internet *(CSAAI)***. Esta asignatura me introdujo en el mundo de las páginas web con lenguajes como JavaScript, HTML5 y CSS. Gracias a ella, he podido visualizar los resultados de mi TFG de forma más amigable.

Además de estas asignaturas destacadas, todas las demás impartidas en el grado han proporcionado su granito de arena para poder hacerme llegar hasta aquí.

7.3. Lecciones aprendidas

En este proyecto de fin de grado, he podido adquirir las siguientes lecciones:

1. La mejora del conocimiento de Python. Este proyecto, al estar orientado al análisis de código de Python, me ha ayudado a mejorar mi nivel y mi fluidez programando en este lenguaje. Destacar también, el módulo de *ast* me ha ayudado a entender mejor su estructura.

2. El uso de la biblioteca *pandas*, la cual me ha ayudado a poder realizar mis primeros análisis de datos, permitiéndome así, recoger datos de encuestas de *Google Forms* y obtener unos resultados estadísticos.

3. El lenguaje LaTeX aprendido en la realización de esta memoria, ya que hasta el momento no lo había utilizado.

4. He mejorado mi capacidad de organización y planificación en un proyecto, aprendiendo así además a ser más autosuficiente.

7.4. Limitaciones

No todos los problemas en este proyecto han sido posibles abordarlos, ya que como en todo trabajo se encuentran ciertas limitaciones. Algunas de las cuales son las siguientes:

- Es fácil engañar a la herramienta, debido a que normalmente cuando se necesita ayuda se recurre a búsquedas por internet provocando así *copy & paste* de código. Este código copiado, hay veces que los usuarios no se detienen a pensar cual es el funcionamiento exacto de ese código, comprender la estructura implementada, etc. De esta manera, provoca que usuarios con un nivel básico como puede ser un A2, tengan elementos aislados que corresponden a un nivel mayor como puede ser un C1 o C2.

- Es difícil establecer una versión consolidada de niveles CEFRL asociada al lenguaje *Python*. Como ya se vio en la sección 5.4, se elaboraron encuestas para poder obtener una versión final, pero fue imposible. Cada usuario tiene una opinión distinta según su nivel y su experiencia en el uso de los elementos. De esta manera, usuarios que no hayan visto un elemento, tenderán a asignarle un nivel mayor, o incluso usuarios con un nivel C2, les cuesta asignar un nivel C2 a un elemento debido a que para ellos es todo *'más fácil'*. Por lo que se podría decir, que los formularios no serían la mejor opción para obtener la *configuración vanilla*, ya que los usuarios que votan no tienen contexto, por ello, otra posibilidad hubiera sido hacer entrevistas a cada usuario.

- La herramienta solo puede analizar código de Python3 y que sea sintácticamente correcto, debido a que el módulo de *ast* nos establece esa limitación. De esta forma, cada vez que analiza algún código que contenga una sintaxis errónea, como puede ser código de una versión distinta o simplemente un error sintáctico, te avisará por la shell de que ese fichero analizado contiene código mal escrito. Por ello, la herramienta no podrá finalizar su análisis, y pasará al siguiente fichero. Esto se tuvo que mitigar con una excepción.

7.5. Trabajos futuros

Esta herramienta solo es una primera idea de lo que podría llegar a ser. Por lo que se le puede añadir una gran variedad de mejoras y funcionalidades. Algunas de las ideas son las siguientes:

- El lenguaje de Python y la combinación de sus elementos son infinitos, por ello se puede ampliar los elementos a analizar a los propuestos en este TFG.
- Adaptar el programa para que el módulo *ast* te permita analizar códigos de Python de versiones anteriores. Esto se podría realizar implementando un analizador propio.
- Mostrar en la visualización de resultados mediante páginas web, que elementos se pueden usar para mejorar el nivel de programación en Python.

Apéndice A

Anexo A

A.1. Niveles del proyecto CEFRL

A.2. Primera versión de niveles

A.3. Segunda versión de niveles

A.4. Configuración *'vanilla'*

		A1 Basic User	A2 Basic User
Writing	Writing code	I can produce a correct implementation for a simple function, given a well-defined specification of desired behavior and interface, without help from others.	I can determine a suitable interface and produce a correct implementation, given a loose specification for a simple function, without help from others. I can break down a complex function specification in smaller functions.
	Refactoring	I can adapt my code when I receive small changes in its specification without rewriting it entirely, provided I know the change is incremental. I can change my own code given detailed instructions from a more experienced programmer.	I can determine myself whether a small change in specification is incremental or requires a large refactoring. I can change my own code given loose instructions from a more experienced programmer.
	Embedding in a larger system	I know the entry and termination points in the code I write. I can use the main I/O channels of my language to input and print simple text and numbers.	I am familiar with recommended mechanisms to accept program options/parameters from the execution environment and signal errors, and use them in the code I write.
Understanding	Reusing code	I can assemble program fragments by renaming variables until the whole becomes coherent and compatible with my goal.	Given a library of mostly pure functions and detailed API documentation, I can reuse this library productively in my code.
	Explaining / Discussing code	I can read the code I wrote and explain what I intend it to mean to someone more experienced than me.	I can read code from someone of a similar or lower level than me and explain what it means. I can recognize and explain simple mismatches between specification and implementation in my code or code from someone at the same level as me or lower.
Interacting	Exploring, self-learning	I can distinguish between a command prompt at a shell and an input prompt for a program run from this shell. I can follow online tutorials without help and reach the prescribed outcome. I can search for the text of common error messages and adapt the experience of other people to my need.	I can distinguish between features general to a language and features specific to a particular language implementation. I can read the text of error messages and understand what they mean without external help.
	Mastery of the environment	I can use a common programming environment and follow common workflows step-by-step to test/run a program.	I can integrate my source files in a programming environment that automates large portions of my programming workflow. I use version control to track my progress and roll back from unsuccessful changes.
	Troubleshooting	I can distinguish between correct and incorrect output in my own programs. I am familiar with the etiquette for asking help from experts in my domain.	I can reliably distinguish between incorrect output due to incorrect input, from incorrect output due to program error. I can narrow down the location of a program error in a complex program to a single module or function. I can isolate and fix Bohr bugs in my own code.

Figura A.1: Tabla de niveles del proyecto CEFRL (I).

B1 Intermediate User	B2 Intermediate User
I can estimate the space and time costs of my code during execution. I can empirically compare different implementations of the same function specification using well-defined metrics, including execution time and memory footprint. I express invariants in my code using preconditions, assertions and post-conditions. I use stubs to gain flexibility on implementation order.	I use typing and interfaces deliberately and productively to structure and plan ahead my coding activity. I can design and implement entire programs myself given well-defined specifications on external input and output. I systematically attempt to generalize functions to increase their reusability.
I can derive a refactoring strategy on my own code, given relatively small changes in specifications. I can change other people's code given precise instructions from a person already familiar with the code.	I can predict accurately the effort needed to adapt my own code base to a new specification. I can follow an existing refactoring strategy on someone else's code. I can take full responsibility for the integration of someone else's patch onto my own code.
I can delegate functions to an external process at run-time. I know how to productively use streaming and buffering to work on large data sets and use them in my code. I am familiar with the notion of locality and use it to tailor my implementations.	I am familiar with at least one API for bi-directional communication with other run-time processes. I can write client code for simple Internet protocols. I am familiar with the most common packaging and redistribution requirements of at least one platform and use them in my own projects. I can use predetermined programming patterns to exploit platform parallelism productively in my code.
I can recognize when existing code requires a particular overall architecture for reuse (e.g. an event loop). I can adapt my own code in advance to the requirements of multiple separate libraries that I plan to reuse.	I can recognize and extract reusable components from a larger code base for my own use, even when the original author did not envision reusability. I can package, document and distribute a software library for others to reuse. I can interface stateless code from different programming languages.
I can show and explain code fragments I write in either imperative or declarative style to someone else who knows a different programming language where the same style is prevalent, so that this person can reproduce the same functionality in their language of choice.	I can explain my data structures, algorithms and architecture patterns to someone else using the standard terms in my domain, without reference to my code.
I can read the reference documentation for the language(s) or API I use, and refer to it to clarify my understanding of arbitrary code fragments. I can understand the general concepts in articles or presentations by experts. I can track and determine who is responsible for an arbitrary code fragment in a system I use or develop for.	I can infer the abstract operating model of an API or library from its interface, without access to documentation, and write small test programs to test if my model is accurate. I can recognize when a reference documentation for a language or API is incomplete or contradictory with a reference implementation.
I express and use dependency tracking in my programming environment to avoid unnecessary (re)processing in my development cycles. I can use different development branches in version control for different programming tasks.	I use different workflows for different programming assignments, with different trade-offs between initial set-up overhead and long-term maintenance overhead. I can enter the environment of someone else at my level or below and make code contributions there with minimal training.
I can translate human knowledge or specifications about invariants into assertions or type constraints in my own code. I can inspect the run-time state of a program to check it matches known invariant. I write and use unit tests where applicable.	I can reduce a program error to the simplest program that demonstrates the same error. I have one or more working strategy to track and fix heisenbugs in code that I can understand. I write and use regression tests for code that I work with directly.

Figura A.2: Tabla de niveles del proyecto CEFRL (II).

C1 Proficient User	C2 Proficient User
I can systematically recognize inconsistent or conflicting requirements in specifications. I can break down a complex program architecture in smaller components that can be implemented separately, including by other people. I can us existing (E)DSLs or metaprogramming patterns to increase my productivity.	I can reliably recognize when under-specification is intentional or not. I can exploit under-specification to increase my productivity in non-trivial ways. I can devise new (E)DSLs or create new metaprogramming patterns to increase my productivity and that of other programmers.
I can reverse-engineer someone else's code base with help from the original specification, and predict accurately the effort needed to adapt it to a new specification.	I can reverse-engineer someone else's code base without original specification, and predict accurately the effort needed to adapt it to a new specification.
I can implement both client and server software for arbitrary protocol specifications. I can quantify accurately the time and space overheads of different communication mechanisms (e.g., syscalls, pipes, sockets). I am familiar with hardware architectures and can predict how sequential programs will behave when changing the underlying hardware. I can estimate the scalability of parallel code fragments on a given platform.	I am familiar with most software architectures in use with systems I develop for. I can work together with system architects to mutually optimize my own software architecture with the overall system architecture. I am familiar with most design and operational cost/benefit trade-offs in systems that I develop for.
I can systematically remove constraints from existing code that are not mandated by specification, to maximize its generality. I can read and understand code that uses APIs most common in my domain without help from their documentation. I can interface code from different programming languages with distinct operational semantics.	I can discover and reliably exploit undocumented/unintended behavior of any code written in a language I understand, including code that I did not write myself.
I can gauge the expertise level of my audience and change the way I talk to them accordingly. I can recognize when an explanation is overly or insufficiently detailed for a given audience, and give feedback accordingly.	I can take part effortlessly in any conversation or discussion about the language(s) I use, and have a good familiarity with idiomatic constructs. I can come up spontaneously with correct and demonstrative code examples for all concepts I need to share with others.
I am able to read and understand most expert literature applicable to the languages I use. I am able to recognize when an academic innovation is applicable to my domain and adapt it for use in my projects.	I can recognize and expose tacit assumptions in expert literature in my domain. I can reliably recognize when the narrative or description of a programming achievement is false or misleading, without testing explicitly.
I modify my programming environment to tailor it to my personal style, and can quantify how these changes impact my productivity. I can productively use the preferred programming environments of at least 80% of all programmers at my level or below.	I can reliably recognize and quantify friction between other programmers and their programming environment. I can measurably improve the productivity of my peers by helping them tailor their environment to their personal style.
I can devise systematic strategies to track and fix mandelbugs in code that I can understand. I can recognize a hardware bug in a system driven mostly by software I designed.	I can track and attribute responsibility accurately for most unexpected/undesired behaviors in systems that I develop for. I can track and isolate hardware bugs in systems where I have access to all software sources.

Figura A.3: Tabla de niveles del proyecto CEFRL (III).

	Class Element	Level
1	Simple List	A1
2	Nested List	A2
3	List Dictionary	B1
4	Simple List Comprehension	C1
5	Nested List Comprehension	C2
6	List Comprehension with If statements	C1
7	Simple Dictionary	A2
8	Nested Dictionary	B1
9	Dictionary List	B1
10	Dictionary of dictionary lists	B2
11	Simple Dict Comprehension	C1
12	Dictionary Comprehension with If statements	C2
13	Dictionary Comprehension with if expression (If-Else)	C2
14	Nested Dict Comprehension	C2
15	Simple Tuple	A1
16	Nested Tuple	A2
17	Files --> 'open' call function	A2
18	Files --> 'write' call function	A2
19	Files --> 'writelines' call function	A2
20	Files --> 'read' call function	A2
21	Files --> 'readline' call function	A2
22	Print	A1
23	Simple Assignment	A1
24	Assigment with sum (total = total + 1)	A1
25	Simplified incremental Assignment (+=)	A2
26	Simple If statements	A1
27	If statements expresion (else)	B1
28	If statements using → __name__ == '__main__'	B2
29	'break' statement	B1
30	'continue' statement	B1

Figura A.4: Primera versión de niveles (I).

31	'pass' statement	B1
32	Simple While Loop	A1
33	While with Else Loop	A2
34	Simple For Loop	A1
35	Nested For Loop	A2
36	For Loop with Tuple as name.	A2
37	For Loop with List to iterate.	A2
38	For Loop with Tuple to iterate.	A2
39	'range' call function	A2
40	'zip' call function	C2
41	'map' call function	C2
42	'enumerate' call function	C2
43	Function with simple argument(s)	A1
44	Function with Default argument	A2
45	Function with * argument	B1
46	Function with ** argument	B1
47	Function with Keyword-Only argument	B1
48	Recursive Functions	B2
49	Return	A1
50	Lambda	B1
51	Generator Functions (yield)	C1
52	Generator Expressions	C1
53	Import	A2
54	From	A2
55	Relative From	B1
56	From with * statements	B1
57	'as' extension	B1
58	Using the struct module	C1
59	Using the pickle module	C1
60	Using the shelve module	C1

Figura A.5: Primera versión de niveles (II).

60	Using the shelve module	C1
61	Using the dbm module	C1
62	Using the re module	C1
63	Using the importlib module	C1
64	Simple Class	B1
65	Inherited Class	B1
66	Using the constructor method --> __init__	B1
67	Descriptors	C1
68	Class Properties	C1
69	Private Methods or Attributes of the class	C2
70	Class Method	C1
71	Static Method	C1
72	Decorator Function	C2
73	Decorator Class	C2
74	Metaclass (3.X) created with --> __new__	C2
75	Metaclass created in the class header --> 'metaclass = '	C2
76	Metaclass (2.X) created as attribute with --> __metaclass__	C2
77	Super function	C2
78	Attribute statements __slots__	C1
79	Simple Attribute	A2
80	Special Class Attribute __class__	B2
81	Special Class Attribute __dict__	B2
82	'try/except' exception	B1
83	'try/else/except' exception	B1
84	'try/try' exception	B1
85	'try/finally' exception	B2
86	'try/except/finally' exception	B2
87	'try/except/else/finally' exception	B2
88	'raise' exception	B1
89	'assert' exception	B1
90	With	B1

Figura A.6: Primera versión de niveles (III).

	Class Element	Level
1	Simple List	A1
2	Nested List	A2
3	List Dictionary	A1
4	Simple List Comprehension	B2
5	Nested List Comprehension	C1
6	List Comprehension with If statements	B2
7	Simple Dictionary	A1
8	Nested Dictionary	A2
9	Dictionary List	A2
10	Dictionary of dictionary lists	B1
11	Simple Dict Comprehension	B2
12	Dictionary Comprehension with If statements	B2
13	Dictionary Comprehension with if expression (If-Else)	B2
14	Nested Dict Comprehension	C1
15	Simple Tuple	A1
16	Nested Tuple	A2
17	Files --> 'open' call function	A1
18	Files --> 'write' call function	A1
19	Files --> 'writelines' call function	A1
20	Files --> 'read' call function	A1
21	Files --> 'readline' call function	A1
22	Print	A1
23	Simple Assignment	A1
24	Assigment with sum (total = total + 1)	A1
25	Simplified incremental Assignment (+=)	A2
26	Simple If statements	A1
27	If statements expresion (else)	A1
28	If statements using → __name__ == '__main__'	A2
29	'break' statement	A2
30	'continue' statement	A2

Figura A.7: Segunda versión de niveles (I).

31	'pass' statement	B1
32	Simple While Loop	A1
33	While with Else Loop	B1
34	Simple For Loop	A1
35	Nested For Loop	A2
36	For Loop with Tuple as name.	A1
37	For Loop with List to iterate.	A1
38	For Loop with Tuple to iterate.	A1
39	'range' call function	A2
40	'zip' call function	B2
41	'map' call function	B2
42	'enumerate' call function	B1
43	Function with simple argument(s)	A2
44	Function with Default argument	B1
45	Function with * argument	B2
46	Function with ** argument	B2
47	Function with Keyword-Only argument	B2
48	Recursive Functions	B1
49	Return	A2
50	Lambda	C1
51	Generator Functions (yield)	C1
52	Generator Expressions	C1
53	Import	A2
54	From	A2
55	Relative From	A2
56	From with * statements	A2
57	'as' extension	A2
58	Using the struct module	B2
59	Using the pickle module	B2
60	Using the shelve module	B2

Figura A.8: Segunda versión de niveles (II).

60	Using the shelve module	B2
61	Using the dbm module	B2
62	Using the re module	B2
63	Using the importlib module	B2
64	Simple Class	B1
65	Inherited Class	B1
66	Using the constructor method --> __init__	B1
67	Descriptors	B2
68	Class Properties	B2
69	Private Methods or Attributes of the class	B2
70	Class Method	C1
71	Static Method	C1
72	Decorator Function	C1
73	Decorator Class	C2
74	Metaclass (3.X) created with --> __new__	C2
75	Metaclass created in the class header --> 'metaclass = '	C2
76	Metaclass (2.X) created as attribute with --> __metaclass__	C2
77	Super function	B2
78	Attribute statements __slots__	C1
79	Simple Attribute	B1
80	Special Class Attribute __class__	B2
81	Special Class Attribute __dict__	B2
82	'try/except' exception	A2
83	'try/else/except' exception	B1
84	'try/try' exception	A2
85	'try/finally' exception	B1
86	'try/except/finally' exception	B1
87	'try/except/else/finally' exception	B1
88	'raise' exception	B1
89	'assert' exception	B1
90	With	A2

Figura A.9: Segunda versión de niveles (III).

	Class element	Level	Standard Deviation
1	Simple List	A1	0.4879500364742666
2	Nested List	B1	0.5345224838248488
3	List Dictionary	A2	0.7867957924694432
4	Simple List Comprehension	B2	1.4142135623730951
5	Nested List Comprehension	C1	1.3801311186847085
6	List Comprehension with If statements	B2	1.0690449676496976
7	Simple Dictionary	A1	0.5345224838248488
8	Nested Dictionary	B1	0.5345224838248488
9	Dictionary List	B1	0.9511897312113419
10	Dictionary of dictionary lists	B1	0.7559289460184544
11	Simple Dict Comprehension	B2	1.3451854182690985
12	Dictionary Comprehension with If statements	B2	1.3451854182690985
13	Dictionary Comprehension with if expression (If-Else)	B2	1.3451854182690985
14	Nested Dict Comprehension	B2	1.4142135623730951
15	Simple Tuple	A2	0.4879500364742666
16	Nested Tuple	A2	0.5345224838248488
17	Files --> 'open' call function	A2	0.9511897312113419
18	Files --> 'write' call function	A2	0.5773502691896257
19	Files --> 'writelines' call function	A2	0.9511897312113419
20	Files --> 'read' call function	A2	0.6900655593423543
21	Files --> 'readline' call function	A2	0.6900655593423543
22	Print	A1	0.3779644730092272
23	Simple Assignment	A1	0.0
24	Assigment with sum (total = total + 1)	A1	0.3779644730092272
25	Simplified incremental Assignment (+=)	A2	0.8997354108424374
26	Simple If statements	A1	0.0
27	If statements expresion (else)	A1	0.7867957924694432
28	If statements using → __name__ == '__main__'	B2	1.2724180205607036
29	'break' statement	A1	0.8997354108424374
30	'continue' statement	A2	1.3801311186847085

Figura A.10: Configuración vanilla (I).

31	'pass' statement	B1	1.1338934190276817
32	Simple While Loop	A1	0.3779644730092272
33	While with Else Loop	A2	1.2724180205607036
34	Simple For Loop	A1	0.3779644730092272
35	Nested For Loop	A2	0.3779644730092272
36	For Loop with Tuple as name.	B1	0.9759000729485332
37	For Loop with List to iterate.	A1	0.8997354108424374
38	For Loop with Tuple to iterate.	A2	1.1338934190276817
39	'range' call function	A2	1.2724180205607036
40	'zip' call function	B1	1.7994708216848747
41	'map' call function	B1	1.9023794624226837
42	'enumerate' call function	B1	1.6035674514745464
43	Function with simple argument(s)	A1	0.5345224838248488
44	Function with Default argument	B1	0.5345224838248488
45	Function with * argument	B2	1.7182493859684491
46	Function with ** argument	B2	1.6035674514745462
47	Function with Keyword-Only argument	B1	0.6900655593423543
48	Recursive Functions	B1	0.6900655593423543
49	Return	A1	0.5345224838248488
50	Lambda	B2	0.816496580927726
51	Generator Functions (yield)	C1	1.3972762620115438
52	Generator Expressions	C1	1.2535663410560174
53	Import	A1	0.5345224838248488
54	From	A2	0.8997354108424374
55	Relative From	A2	1.4638501094227998
56	From with * statements	B1	1.2724180205607036
57	'as' extension	B1	0.7559289460184544
58	Using the struct module	C2	1.4638501094227998
59	Using the pickle module	Don't know	2.627691364061218
60	Using the shelve module	Don't know	2.627691364061218

Figura A.11: Configuración vanilla (II).

61	Using the dbm module	C1	2.4784787961282104
62	Using the re module	C1	2.0586634591635513
63	Using the importlib module	C1	2.309401076758503
64	Simple Class	B1	0.8997354108424374
65	Inherited Class	B1	0.6900655593423543
66	Using the constructor method --> __init__	B1	0.9759000729485332
67	Descriptors	C1	1.1126972805283737
68	Class Properties	C1	1.3972762620115438
69	Private Methods or Attributes of the class	B2	1.5118578920369088
70	Class Method	B1	1.1126972805283737
71	Static Method	C1	1.4142135623730951
72	Decorator Function	C1	1.1338934190276817
73	Decorator Class	C1	1.0690449676496976
74	Metaclass (3.X) created with --> __new__	C2	1.4960264830861913
75	Metaclass created in the class header --> 'metaclass = '	C2	2.2253945610567474
76	Metaclass (2.X) created as attribute with --> __metaclass__	C2	2.3603873774083293
77	Super function	B2	1.2909944487358056
78	Attribute statements __slots__	C2	1.6035674514745464
79	Simple Attribute	A2	0.816496580927726
80	Special Class Attribute __class__	B2	1.2909944487358056
81	Special Class Attribute __dict__	C2	1.4960264830861913
82	'try/except' exception	B1	0.7559289460184544
83	'try/else/except' exception	B1	0.6900655593423543
84	'try/try' exception	B1	0.5345224838248488
85	'try/finally' exception	B2	0.7559289460184544
86	'try/except/finally' exception	B2	0.8997354108424374
87	'try/except/else/finally' exception	B1	0.9759000729485332
88	'raise' exception	B1	0.8997354108424374
89	'assert' exception	B1	1.3972762620115438
90	With	A2	1.3801311186847085

Figura A.12: Configuración vanilla (III).

Apéndice B

Anexo B

B.1. Comparativa de clases y niveles

```
Ana:  Nested List: A2
Grex: Nested List: A2

Ana:  List Dictionary: B1
Grex: List Dictionary: A1

Ana:  Simple List Comprehension: C1
Grex: Simple List Comprehension: B2

Ana:  Nested List Comprehension: C2
Grex: Nested List Comprehension: C1

Ana:  List Comprehension with If statements: C1
Grex: List Comprehension with If statements: B2

Ana:  Simple Dictionary: A2
Grex: Simple Dictionary: A1

Ana:  Nested Dictionary: B1
```

```
Grex: Nested Dictionary: A2

Ana:  Dictionary List: B1
Grex: Dictionary List: A2

Ana:  Dictionary of dictionary lists: B2
Grex: Dictionary of dictionary lists: B1

Ana:  Simple Dict Comprehension: C1
Grex: Simple Dict Comprehension: B2

Ana:  Dictionary Comprehension with If statements: C2
Grex: Dictionary Comprehension with If statements: B2

Ana:  Dictionary Comprehension with if expression: C2
Grex: Dictionary Comprehension with if expression: B2

Ana:  Nested Dict Comprehension: C2
Grex: Nested Dict Comprehension: C1

Ana:  Simple Tuple: A1
Grex: Simple Tuple: A1

Ana:  Nested Tuple: A2
Grex: Nested Tuple: A2

Ana:  Files -->  'open' call function: A2
Grex: Files -->  'open' call function: A1

Ana:  Files -->  'write' call function: A2
Grex: Files -->  'write' call function: A1
```

```
Ana:  Files -->  'writelines' call function: A2
Grex: Files -->  'writelines' call function: A1

Ana:  Files -->  'read' call function: A2
Grex: Files -->  'read' call function: A1

Ana:  Files -->  'readline' call function: A2
Grex: Files -->  'readline' call function: A1

Ana:  Print: A1
Grex: Print: A1

Ana:  Simple Assignment: A1
Grex: Simple Assignment: A1

Ana:  Assignment with sum (total = total + 1) : A1
Grex: Assignment with sum (total = total + 1) : A1

Ana:  Simplified incremental Assignment (+=): A2
Grex: Simplified incremental Assignment (+=): A2

Ana:  Simple If statements: A1
Grex: Simple If statements: A1

Ana:  If statements expression (else): B1
Grex: If statements expresión (else): A1

Ana:  If statements using → __name__ == '__main__': B2
Grex: If statements using → __name__ == '__main__': A2
```

```
Ana:  'break' statement: B1
Grex: 'break' statement: A2

Ana:  'continue' statement: B1
Grex: 'continue' statement: A2

Ana:  'pass' statement: B1
Grex: 'pass' statement: B1

Ana:  Simple While Loop: A1
Grex: Simple While Loop: A1

Ana:  While with Else Loop: A2
Grex: While with Else Loop: B1

Ana:  Simple For Loop: A1
Grex: Simple For Loop: A1

Ana:   Nested For Loop: A2
Grex:  Nested For Loop: A2

Ana:  For Loop with Tuple as name.: A2
Grex: For Loop with Tuple as name.: A1

Ana:  For Loop with List to iterate.: A2
Grex: For Loop with List to iterate.: A1

Ana:  For Loop with Tuple to iterate.: A2
Grex: For Loop with Tuple to iterate.: A1

Ana:  'range' call function: A2
```

```
Grex: 'range' call function: A2

Ana:  'zip' call function: C2
Grex: 'zip' call function: B2

Ana:  'map' call function: C2
Grex: 'map' call function: B2

Ana:  'enumerate' call function: C2
Grex: 'enumerate' call function: B1

Ana:  Function with simple argument(s): A1
Grex: Function with simple argument(s): A2

Ana:  Function with Default argument: A2
Grex: Function with Default argument: B1

Ana:  Function with * argument: B1
Grex: Function with * argument: B2

Ana:  Function with ** argument: B1
Grex: Function with ** argument: B2

Ana:  Function with Keyword-Only argument: B1
Grex: Function with Keyword-Only argument: B2

Ana:  Recursive Functions: B2
Grex: Recursive Functions: B1

Ana:  Return: A1
Grex: Return: A2
```

```
Ana:  Lambda: B1
Grex: Lambda: C1

Ana:  Generator Functions (yield): C1
Grex: Generator Functions (yield): C1

Ana:  Generator Expressions: C1
Grex: Generator Expressions: C1

Ana:  Import: A2
Grex: Import: A2

Ana:  From: A2
Grex: From: A2

Ana:  Relative From: B1
Grex: Relative From: A2

Ana:  From with * statements: B1
Grex: From with * statements: A2

Ana:  'as' extension: B1
Grex: 'as' extension: A2

Ana:  Simple Class: B1
Grex: Simple Class: B1

Ana:  Inherited Class: B1
Grex: Inherited Class: B1
```

```
Ana:  Using the constructor method --> __init__: B1
Grex: Using the constructor method --> __init__: B1

Ana:  Descriptors: C1
Grex: Descriptors: B2

Ana:  Class Properties: C1
Grex: Class Properties: B2

Ana:  Private Methods or Attributes of the class: C2
Grex: Private Methods or Attributes of the class: B2

Ana:  Class Method: C1
Grex: Class Method: C1

Ana:  Static Method: C1
Grex: Static Method: C1

Ana:  Decorator Function: C2
Grex: Decorator Function: C1

Ana:  Decorator Class: C2
Grex: Decorator Class: C2

Ana:  Metaclass (3.X) created with -->
      __new__: C2
Grex: Metaclass (3.X) created with -->
      __new__: C2

Ana:  Metaclass created in the class header -->
      'metaclass = ': C2
```

```
Grex: Metaclass created in the class header -->
      'metaclass = ': C2

Ana:  Metaclass (2.X) created as attribute with -->
      __metaclass__: C2
Grex: Metaclass (2.X) created as attribute with -->
      __metaclass__: C2

Ana:  Super function: C2
Grex: Super function: B2

Ana:  Attribute  statements __slots__: C1
Grex: Attribute  statements __slots__: C1

Ana:  Simple Attribute: A2
Grex: Simple Attribute: B1

Ana:  Special Class Attribute __class__: B2
Grex: Special Class Attribute __class__: B2

Ana:  Special Class Attribute __dict__: B2
Grex: Special Class Attribute __dict__: B2

Ana:  'try/except' exception: B1
Grex: 'try/except' exception: A2

Ana:  'try/else/except' exception: B1
Grex: 'try/else/except' exception: B1

Ana:  'try/try' exception: B1
Grex: 'try/try' exception: A2
```

```
Ana:  'try/finally' exception: B2
Grex: 'try/finally' exception: B1

Ana:  'try/except/finally' exception: B2
Grex: 'try/except/finally' exception: B1

Ana:  'try/except/else/finally' exception: B2
Grex: 'try/except/else/finally' exception: B1

Ana:  `raise' exception: B1
Grex: `raise' exception: B1

Ana:  'assert' exception: B1
Grex: 'assert' exception: B1

Ana:  With: B1
Grex: With: A2
```

B.2. Comparativa de numero de elementos en cada nivel

```
Ana:
Elements of A1 level: 9
Elements of A2 level: 19
Elements of B1 level: 23
Elements of B2 level: 8
Elements of C1 level: 10
Elements of C2 level: 14

Grex:
```

```
Elements of A1 level: 18
Elements of A2 level: 20
Elements of B1 level: 16
Elements of B2 level: 16
Elements of C1 level: 9
Elements of C2 level: 4
```

Bibliografía

[1] Jupyter notebook.
https://jupyter.org/.

[2] Pandas.
https://pandas.pydata.org/.

[3] C. V. Alexandru, J. J. Merchante, S. Panichella, S. Proksch, H. C. Gall, and G. Robles. On the usage of pythonic idioms. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 1–11, 2018.

[4] L. V. d. P. Jaime Cerda L. Evaluación de la concordancia inter-observador en investigación pediátrica: Coeficiente de kappa. *Revista Chilena de Pediatría*, 79:54–58, 2008.

[5] K. G. Landis J. The measurement of observer agreement for categorical data. *Biometrics*, 33:159–74, 1977.

[6] M. Lutz's. *Learning Python, 5th Edition*. O'Reilly, 2013.

[7] V. H. Torres Gordillo, Juan Jesús; Perera Rodríguez. Cálculo de la fiablidad y concordancia entre codificadores de un sistema de categorías para el estudio del foro online en e-learning. *Revista de Investigación Educativa*, 27:89–103, 2009.

[8] R. 'kena' Poss. How good are you at programming? a cefr-like approach to measure programming proficiency, June 2014.
https://dr-knz.net/programming-levels.html.