

UNIVERSIDAD DE OVIEDO

**TRABAJO FIN DE GRADO**

Sistema para la extracción y análisis de construcciones sintácticas de código Python

**Abel Busto Dopazo**

**DIRECTORES**

**FRANCISCO ORTIN SOLER**

**MIGUEL GARCÍA RODRÍGUEZ**

*Proyecto presentado en cumplimiento de los requisitos*

*para el Grado en Ingeniería Informática del Software en el*

[*Computational Reflection Research Group*](http://www.reflection.uniovi.es/)

Departamento de Informática

Resumen

El objetivo principal de este proyecto de investigación es encontrar un mecanismo general para la extracción de construcciones sintácticas de código Python para su posterior análisis y explotación. La representación sintáctica de programas de ordenador suele llevarse a cabo con estructuras conocidas como Árboles de Sintaxis Abstracta (*AST*s). La extracción de estas estructuras es un paso necesario para poder analizar las construcciones sintácticas utilizadas por los programadores. Esta información puede ser utilizada en diferentes campos de aplicación, como la enseñanza de la programación, la implementación de *IDE*s y la documentación de patrones. Para poder trabajar con grandes volúmenes de programas, es necesario almacenar esas estructuras en bases de datos. Elegiremos para ello el modelo relacional por su madurez y eficiencia. El principal problema es que la información sintáctica de los programas se representa jerárquicamente mediante estructuras de árbol o grafo. Por lo tanto, es necesario representar los distintos patrones sintácticos (clases, métodos, sentencias, expresiones, etc.) en formato de tabla. Para cada construcción sintáctica, se identifica su contexto mediante información del propio nodo, de su nodo padre, de los nodos hijos y de sus niveles de profundidad y anchura. Así se aúna información local y global de cada nodo. El primer objetivo de este trabajo es la extracción del *AST* de cualquier programa Python. Para llevar a cabo este objetivo, el sistema propuesto modifica la salida del módulo *AST* de la Librería Estándar de Python (*Python Standar Library*) para ampliar la información sintáctica proporcionada por su *AST*. El segundo objetivo del presente proyecto es la transformación de esos *AST*s en un modelo relacional que permita almacenar grandes volúmenes de código, pudiéndolos consultar de un modo eficiente. El sistema implementa un mecanismo para traducir los *AST*s obtenidos en vectores n-dimensionales que pueden ser almacenados en forma de tabla. De este modo, será posible analizar las distintas construcciones sintácticas empleadas por los programadores de numerosos proyectos reales. La información obtenida puede ser usada en diferentes escenarios de aplicación, como la enseñanza de la programación, la implementación de *IDE*s y la documentación de patrones de programación. A modo de ejemplo, en este trabajo se hace un análisis detallado sobre las anomalías encontradas en el uso que hacen los programadores Python de las diferentes construcciones sintácticas en las que se divide su código.

Palabras clave

Lenguajes de programación, construcciones sintácticas, Árboles de Sintaxis Abstracta, Python

Abstract

TODO

Keywords

TODO

Agradecimientos

TODO

Índice de contenido

[1 Introducción 9](#_Toc168929990)

[2 Trabajo relacionado 10](#_Toc168929991)

[3 Descripción del sistema 11](#_Toc168929992)

[3.1 Extracción de ASTs 12](#_Toc168929993)

[3.2 Generación de tablas 13](#_Toc168929994)

[3.2.1 Programs 14](#_Toc168929995)

[3.2.2 Modules 15](#_Toc168929996)

[3.2.3 Imports 16](#_Toc168929997)

[3.2.4 Class Definitions 17](#_Toc168929998)

[3.2.5 Function Definitions 19](#_Toc168929999)

[3.2.6 Method Definitions 20](#_Toc168930000)

[3.2.7 Statements 21](#_Toc168930001)

[3.2.8 Cases 22](#_Toc168930002)

[3.2.9 Handlers 23](#_Toc168930003)

[3.2.10 Expressions 23](#_Toc168930004)

[3.2.11 Comprehensions 24](#_Toc168930005)

[3.2.12 CallArgs 24](#_Toc168930006)

[3.2.13 FStrings 25](#_Toc168930007)

[3.2.14 Variables 25](#_Toc168930008)

[3.2.15 Vectors 26](#_Toc168930009)

[3.2.16 Parameters 26](#_Toc168930010)

[3.3 Detección de valores atípicos 27](#_Toc168930011)

[4 Metodología 27](#_Toc168930012)

[4.1 Conjunto de Datos 27](#_Toc168930013)

[4.2 Entorno de ejecución 29](#_Toc168930014)

[5 Evaluación 30](#_Toc168930015)

[5.1 Programas 30](#_Toc168930016)

[5.2 Módulos 30](#_Toc168930017)

[5.3 Imports 31](#_Toc168930018)

[5.4 Definiciones de clase 31](#_Toc168930019)

[5.5 Definiciones de funciones 32](#_Toc168930020)

[5.6 Definiciones de métodos 32](#_Toc168930021)

[5.7 Sentencias 33](#_Toc168930022)

[5.8 Cases 33](#_Toc168930023)

[5.9 Handler 34](#_Toc168930024)

[5.10 Expresiones 34](#_Toc168930025)

[5.11 Comprehensions 35](#_Toc168930026)

[5.12 Invocaciones a funciones 35](#_Toc168930027)

[5.13 Cadenas formateadas 35](#_Toc168930028)

[5.14 Variables 35](#_Toc168930029)

[5.15 Vectores 35](#_Toc168930030)

[5.16 Parámetros 35](#_Toc168930031)

[6 Conclusiones y Trabajo Futuro 36](#_Toc168930032)

[6.1 Conclusiones 36](#_Toc168930033)

[6.2 Trabajo Futuro 36](#_Toc168930034)

[7 Planificación y Presupuesto 37](#_Toc168930035)

[7.1 Planificación del proyecto 37](#_Toc168930036)

[7.2 Planificación del proyecto 37](#_Toc168930037)

[7.2.1 Precios por hora 37](#_Toc168930038)

[7.2.2 Precio por unidad de trabajo 38](#_Toc168930039)

[7.2.3 Presupuesto total 41](#_Toc168930040)

[8 Referencias 42](#_Toc168930041)

[9 Anexos 43](#_Toc168930042)

[9.1 Diseño de la Base de Datos 44](#_Toc168930043)

[9.2 Dominio de las características 46](#_Toc168930044)

[9.2.1 StatementCategory 46](#_Toc168930045)

[9.2.2 StatementRole 46](#_Toc168930046)

[9.2.3 ExpressionCategory 46](#_Toc168930047)

[9.2.4 ExpressionRole 46](#_Toc168930048)

[9.3 Resultados Detección de Anomalías 47](#_Toc168930049)

[9.3.1 Programas 47](#_Toc168930050)

[9.3.2 Modulos 48](#_Toc168930051)

[9.3.3 Imports 49](#_Toc168930052)

[9.3.4 Definiciones de clases 50](#_Toc168930053)

[9.3.5 Definiciones de funciones 52](#_Toc168930054)

[9.3.6 Definiciones de métodos 54](#_Toc168930055)

[9.3.7 Sentencias 57](#_Toc168930056)

[9.3.8 Cases 60](#_Toc168930057)

[9.3.9 Handlers 60](#_Toc168930058)

[9.3.10 Expresiones 61](#_Toc168930059)

[9.3.11 Comprehensions 64](#_Toc168930060)

[9.3.12 Invocaciones a funciones 65](#_Toc168930061)

[9.3.13 Cadenas formateadas 65](#_Toc168930062)

[9.3.14 Variables 66](#_Toc168930063)

[9.3.15 Vectores 66](#_Toc168930064)

[9.3.16 Parámetros de declaración de funciones 67](#_Toc168930065)

[9.4 Repositorios GitHub 68](#_Toc168930066)

Lista de Figuras

[Figura 1: Arquitectura del sistema propuesto para la extracción de información sintáctica. 11](#_Toc168930067)

[Figura 2: Relación de clases del AST del PSL y del sistema propuesto. 13](#_Toc168930068)

[Figura 3: Diagrama entidad relación con las tablas de la base de datos 44](#_Toc168930069)

Lista de Tablas

[Tabla 1: Características de programa. 14](#_Toc168930070)

[Tabla 2: Características para el módulo. 15](#_Toc168930071)

[Tabla 3: Características para los imports. 17](#_Toc168930072)

[Tabla 4: Características para la definición de clases. 18](#_Toc168930073)

[Tabla 5: Características para definiciones de funciones. 19](#_Toc168930074)

[Tabla 6: Características de definiciones de métodos. 20](#_Toc168930075)

[Tabla 7: Características de sentencias. 21](#_Toc168930076)

[Tabla 8: Características de los cases. 22](#_Toc168930077)

[Tabla 9: Características de los handlers. 23](#_Toc168930078)

[Tabla 10: Características de expression. 23](#_Toc168930079)

[Tabla 11: Características de comprehensiones. 24](#_Toc168930080)

[Tabla 12: Características de las invocaciones a funciones. 24](#_Toc168930081)

[Tabla 13: Características de las cadenas formateadas. 25](#_Toc168930082)

[Tabla 14: Características de las variables. 25](#_Toc168930083)

[Tabla 15: Características de los vectores. 26](#_Toc168930084)

[Tabla 16: Características de los parámetros. 26](#_Toc168930085)

[Tabla 17: Número de Nodos de los ASTs 28](#_Toc168930086)

[Tabla 18: Planificación del Proyecto 37](#_Toc168930087)

[Tabla 19: Precio por hora investigador 38](#_Toc168930088)

[Tabla 20: Precio por hora programador 38](#_Toc168930089)

[Tabla 21: Precios por unidad de trabajo. Parte 1: Decisión de alcance 38](#_Toc168930090)

[Tabla 22: Precios por unidad de trabajo. Parte 2: Elección de tecnologías 38](#_Toc168930091)

[Tabla 23: Precios por unidad de trabajo. Parte 3: Diseño de la arquitectura 38](#_Toc168930092)

[Tabla 24: Precios por unidad de trabajo. Parte 4: Desarrollo del programa 39](#_Toc168930093)

[Tabla 3425: Precios por unidad de trabajo. Parte 14: Documentación del proyecto 40](#_Toc168930094)

[Tabla 266: Conceptos de los costes indirectos 41](#_Toc168930095)

# Introducción

TODO

De Guillermo

los algoritmos de aprendizaje automático interpretables pueden utilizarse para extraer información de las construcciones sintácticas. Esta información se podría emplear con diferentes objetivos. Por ejemplo, se podrían documentar las construcciones recurrentes escritas por programadores noveles, medios y expertos. Los profesores de asignaturas de programación podrían identificar los patrones de programación recurrentes utilizados por los estudiantes, incluidos los que son propensos a errores, y explicar cómo podrían mejorarse con alternativas mejores (por ejemplo, las construcciones empleadas por los expertos) [15]. Esos patrones también pueden ser útiles en la construcción de decompiladores, entornos de desarrollo integrados (IDEs) y sistemas de tutoría inteligentes (ITS) [16].

La principal contribución de este trabajo es la extracción de construcciones sintácticas de código Python, definiendo técnicas de transformación de ASTs a tablas que nos permitan almacenar y analizar grandes volúmenes de datos de forma eficiente. Para ello:

1. Diseñamos un AST con un grano de detalle de sus entidades mucho más fino que el utilizado por el módulo AST de la Librería Estándar de Python (*Python Standar Library, PSL*) [1], ampliando el nivel de detalle ofrecido por nuestro nuevo diseño del AST.
2. Modificamos el análisis sintáctico que realiza la *PSL* para que cree estos nuevos árboles enriquecidos en lugar de los originales.
3. Definimos un mecanismo de traducción de los ASTs a información tabular.
4. Analizamos y detectamos valores anómalos en los conjuntos de datos generados.

Este trabajo se encuentra organizado de la siguiente manera. El siguiente capítulo describe el trabajo relacionado. El Capítulo 3 presenta la descripción del sistema propuesto y el Capítulo 4 describe la metodología seguida para la evaluación del sistema. En el Capítulo 5 se explican los resultados de los diferentes experimentos, mientras que el Capítulo 6 detalla las conclusiones y el trabajo futuro. Finalmente, el Capítulo 7 detalla la planificación y presupuesto.

# Trabajo relacionado

En el trabajo de Alvaro se realiza un trabajo similar para el lenguaje de programación Java.

En el trabajo [IRJET\_V7I8755-libre.pdf (d1wqtxts1xzle7.cloudfront.net)](https://d1wqtxts1xzle7.cloudfront.net/64732739/IRJET_V7I8755-libre.pdf?1603282304=&response-content-disposition=inline%3B+filename%3DIRJET_Comparative_Analysis_of_Python_and.pdf&Expires=1718117865&Signature=Zjw8-BZirtbzpZvvmIMvWpgQlbgAyO1GFBr8uoFQXXNUGvcLUspUKrCd-VSq9m5mog-94gQhWoYieYNrcKXPT-7aBGLExPMkJWucK-UsIyOtygDi0q-7vephiG39HKz~BOYQ-1kfvaY7jsi13~qUrhxe5zYleFqqGckemfXH55rVXyZ9UWB3c7EKtWMDkOrrtSZ2a3Oif5yHSXWceyhHcEEabMIEgu2L9U3e9yBDKDOkfxPqDpmR~b1AzQJjmsmh9Sy881q2o34b9s81sUgPrF80cqKfdSBWrn8qyPHFikoagv8TZnhfcZwTucwFmcIuj4XDuSvNc3Hr7qbosLH~9w__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA) se habla de la sintaxis de Python, comparada con la Java, en lo que respecta a la sencilles de principiantes para aprender dicho lenguaje.

En el trabajo [Using-Static-Analysis-Tools-for-Analyzing-Student-Behavior-in-an-Introductory-Programming-Course.pdf (researchgate.net)](https://www.researchgate.net/profile/Ibrahim-Albluwi/publication/342347638_Using_Static_Analysis_Tools_for_Analyzing_Student_Behavior_in_an_Introductory_Programming_Course/links/5eef7034299bf1faac691626/Using-Static-Analysis-Tools-for-Analyzing-Student-Behavior-in-an-Introductory-Programming-Course.pdf?_sg%5B0%5D=started_experiment_milestone&origin=journalDetail) se hace un análisis estático del código escrito por principiantes.

En el trabajo [An Empirical Study for Common Language Features Used in Python Projects | IEEE Conference Publication | IEEE Xplore](https://ieeexplore.ieee.org/abstract/document/9425916) se hace un análisis de diferentes patrones Python aplicados a proyectos Python de expertos.

En el trabajo [Comparing python programs using abstract syntax trees (uniandes.edu.co)](https://repositorio.uniandes.edu.co/entities/publication/6a6f3e62-9d31-4d2f-8501-1d92df5cf1a5) se utilizan AST para la comparación 1 a 1 de programas escritos en Python, con el objetivo de evaluar sus similitudes.

En el trabajo [Assessing Developer Expertise from the Statistical Distribution of Programming Syntax Patterns | Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (acm.org)](https://dl.acm.org/doi/abs/10.1145/3463274.3463343) se utilizan patrones sintácticos para evaluar la adecuación de un código a la etiqueta de expert o novato.

# Descripción del sistema

La Figura 1 muestra el diagrama de la arquitectura de nuestro sistema. A continuación, describimos someramente sus elementos para posteriormente profundizar en los mismos. La entrada del sistema es un conjunto de programas Python obtenidos tanto de GitHub como de una asignatura de primer año de programación de un Grado en Ingeniería Informática del Software de la Universidad de Oviedo (Capítulo 4.1), para utilizar código escrito por principiantes y expertos. La salida es un conjunto de datos con las construcciones sintácticas extraídas, su relación con el nivel de experiencia del programador y un informe acerca de los patrones atípicos detectados.



Figura 1: Arquitectura del sistema propuesto para la extracción de información sintáctica.

Lo primero que hacemos es modificar la salida del módulo *AST* de la *Python Standard Library* (*PSL*), aumentando la información sintáctica ofrecida por éste. Este se hace implementando el patrón de diseño *Visitor* [2] que, recorriendo el *AST* original, crea uno nuevo con información más específica. Para ello, rediseñamos el *AST*, añadiendo nuevos tipos de nodos, atributos y relaciones, proporcionando así una información más detallada que la ofrecida por el módulo original (Sección 3.1)

A continuación, se realiza una trasformación de los *AST*s a tablas de un modelo relacional. El sistema identifica siete tipos diferentes de construcciones sintácticas principales: programa, modulo, definición de clase, de función y de método, sentencia y expresión. Además de estas construcciones, el sistema distingue casos concretos de sentencias como *cases* y *handlers*; y de expresiones como *comprehnesion*, invocación, cadena formateada, variable y vector. Por último, también se identifican las construcciones utilizadas para importar módulos y paquetes, así como las utilizadas en la especificación de los parámetros de funciones y métodos. En total el sistema propuesto identifica 16 tipos distintos de construcciones sintácticas. Para cada una de ellas, se genera una tabla en una base de datos relacional, que almacena la información de cada nodo del AST. Se ha llevado a cabo un proceso manual de extracción de características para traducir las estructuras de árbol en tablas (Sección 3.2).

Posteriormente, los datos de las tablas son procesados y filtrados para posteriormente proceder con la detección de anomalías (Sección 3.3). Si las anomalías son debidas a errores de medición (entradas que no debían haber sido consideradas) se eliminan; en caso contrario, se incluyen en un informe de anomalías.

## Extracción de ASTs

Tal y como hemos mencionado anteriormente, la principal estructura de datos para representar sintácticamente un programa es el árbol de sintaxis abstracta (*Abstract Syntax Tree*, *AST*). Cada nodo de un *AST* representa una construcción sintáctica de un programa como una definición de un tipo, una variable, la invocación a un método o una expresión aritmética. Python es un lenguaje de programación interpretado, conocido por su simplicidad y legibilidad, lo que facilita la escritura y el mantenimiento de código [3]. A diferencia de los lenguajes compilados que utilizan representaciones intermedias de código [4], como el *bytecode* en Java o el *Intermediate Language* (*IL*) en .NET, Python ejecuta el código directamente a través de un intérprete. El análisis de las construcciones sintácticas en Python puede ser un reto debido a la falta de un proceso de compilación formal. Para abordar este desafío, se puede utilizar el módulo *AST* de la *Python Standard Library* (*PSL*) [1]. Este módulo proporciona una representación abstracta y jerárquica del código fuente de Python, permitiendo la manipulación de la estructura del código. Es posible descomponer los programas en sus componentes básicos, facilitando así el estudio de las distintas construcciones sintácticas empleadas por los programadores.

A continuación, vamos a explicar el proceso para la modificación del *AST* generado por la *PSL*. Por cada fichero Python, la *PSL* realiza un análisis sintáctico del mismo y genera un *AST*. Mediante el patrón de diseño *Visitor* [2] se recorre el *AST* original para añadir información más detallada sobre la estructura sintáctica del programa, incluyendo nuevas categorías sintácticas, relaciones o atributos. A modo de ejemplo, todas las operaciones binarias en Python son representadas en el *PSL* como instancias de BinOp, dificultando el conocimiento del tipo concreto de operación que representa. Para dar más información acerca de la expresión, hemos añadido nodos que representan operaciones binarias concretas en función del operador: operaciones aritméticas (Arithmetic), potencias (Pow), desplazamiento de bits (Shift) u operaciones lógicas a nivel de bits (BWLogical), entre otras. Esta información extra nos permite identificar que la primera suele ser más habitual entre los programadores noveles, pudiéndose detectar como patrón común a este tipo de programadores. La última es más comúnmente utilizada por programadores expertos. Lo mismo sucede cuando declaramos un valor o un literal. La *PSL* interpreta todos los literales como instancias de una misma entidad Constant. Sin embargo, el sistema propuesto tiene en cuenta el tipo de literal y permite clasificar la expresión con una de las siguientes clases: IntLiteral, FloatLiteral, ComplexLiteral, NoneLiteral, BoolLiteral, StringLiteral o EllipsisLiteral.

Además de las nuevas clases, nuestro nuevo diseño también añade nuevas relaciones para almacenar información sintáctica de interés. Por ejemplo, los nodos de sentencias o expresiones del nuevo *AST* incorporan información relativa al rol que éstos juegan en su construcción sintáctica padre. Así, por ejemplo, una asignación puede jugar dos roles distintos si su padre es una sentencia while, podría ser la condición del bucle o formar parte de su cuerpo. La información que denota cuándo una asignación se utiliza como condición es significativa, por ejemplo, para detectar programadores no principiantes (éstos rara vez usan asignaciones en las condiciones de while). Estas modificaciones nos permiten ampliar el nivel de detalle en comparación con la versión original, donde solamente conoceríamos la clase que define a un nodo y las que definen a sus hijos.

La Figura 2 muestra la diferencia entre las 58 clases del *AST* de la *PSL* [1] y las 80 definidas en el nuevo diseño propuesto.

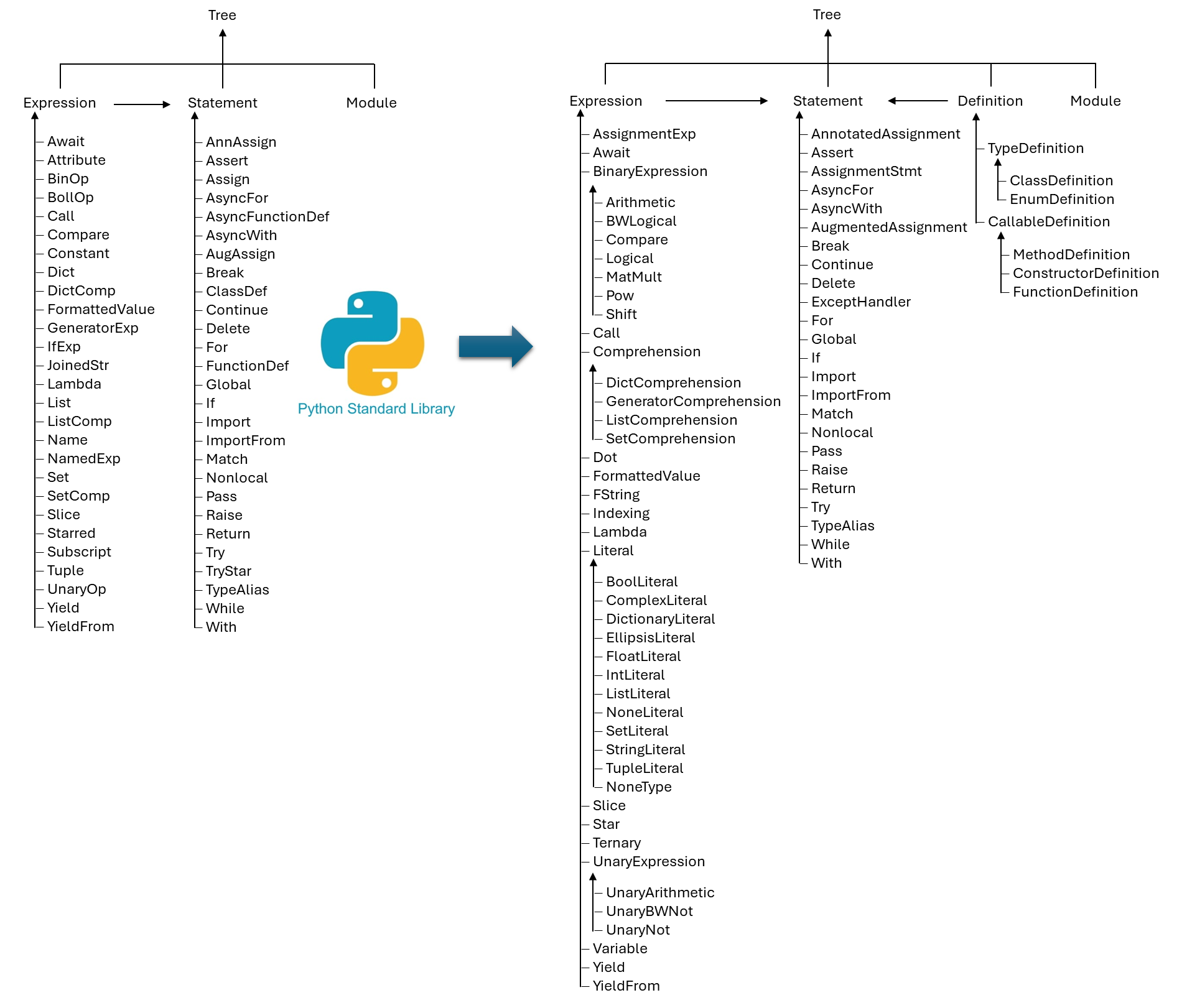


Figura 2: Relación de clases del AST del PSL y del sistema propuesto.

## Generación de tablas

Tal y como mencionamos anteriormente, los algoritmos de minería de datos y aprendizaje automático pueden utilizarse para extraer información valiosa de las construcciones sintácticas. Sin embargo, muchos de estos algoritmos están diseñados para trabajar con datos en formato tabular. Por esta razón, es necesario convertir los *AST*s en tablas para poder aplicar estos algoritmos de manera efectiva. Lo primero es definir los distintos tipos de nodos del *AST* que tengan una estructura común, para poder almacenarlos en la misma tabla. Identificamos los siguientes siete tipos de nodos: *program* (conjunto de directorios y ficheros Python que componen un proyecto independiente), *module* (fichero), *class definition* (definiciones de clases), *function definition* (definición de funciones), *method definition* (definiciones de métodos), *statement* (sentencias) y *expression* (expresiones).

Además de estas entidades principales, usamos *imports* para guardar información resumida de los imports que usa un módulo. La información de los parámetros de las definiciones de funciones y métodos se guarda en *parameters*. También usamos entidades para representar información que solo está presente en el caso de determinadas sentencias. Usamos *case*, para almacenar información exclusiva de las sentencias de tipo Match y *handlers*, para la información relativa a las sentencias Try y TryStar. Por último, almacenamos de forma independiente información específica de algunas expresiones, como *comprehensions* (generadores de listas, diccionarios, tuplas y sets), *callargs* (invocaciones a funciones), *fstring* (cadenas de texto formateadas), *variable* (variables) y *vector* (información relativa a los literales de tipo lista, diccionario, tuplas y sets). Por último, incluimos una tabla *nodes* que sirve para relacionar cada nodo con su nodo padre. Esta tabla contiene únicamente los identificadores del elemento, de su elemento padre y, además, el nombre de la tabla a la que pertenece el elemento padre para facilitar las consultas. Para cada una de estas entidades identificadas, se ha llevado a cabo un proceso manual de extracción y definición de características.

Para realizar la traducción, hacemos uso del patrón de diseño *Visitor* [2]. Recorremos el *AST* obteniendo y almacenando información de cada construcción sintáctica en una tabla correspondiente de la base de datos. Para la recolección de la información se han utilizado dos técnicas. La primera de ellas consiste en propagar información hacia abajo. Un nodo padre, por ejemplo, una expresión trasmite información relativa a sus características como su profundidad o su categoría sintáctica, para que sus nodos hijos puedan almacenar información relativa a su padre. Por otro lado, se trasmite información hacia arriba. Un nodo de definición de método trasmite información, como por ejemplo su número de anotaciones de tipos o el tamaño de su cuerpo, hacia su nodo padre (típicamente una definición de clase) para que en este se pueda almacenar información relativa a las características de sus hijos.

Un ejemplo concreto de este funcionamiento se da entre los FunctionDef y los Statements. El nodo FunctionDef pasa a sus hijos su categoría sintáctica para que estos puedan rellenar la información relativa a su padre (atributo parent). A su vez, los hijos de tipo Statement pasan su categoría sintáctica al nodo FunctionDef para que este pueda llenar la información relativa a la distribución de categorías sintácticas en su cuerpo (atributos expresión\_pct y body\_count).

Además, se ha realizado un sistema de claves internas para poder identificar la estructura de árbol concreta de cada programa. La Figura 4 del Anexo 9.1 muestra el modelo entidad relación con el diseño de la base de datos empleado en el sistema.

Las tablas 1 a 16 muestran toda la información almacenada para todas las construcciones sintácticas. A modo de ejemplo inicial, la Tabla 7 muestra la información almacenada para las sentencias. Además del nombre de la clase de la sentencia (su categoría sintáctica), guardamos la categoría sintáctica de su nodo padre. También, almacenamos el rol que la sentencia juega en el nodo padre. Asimismo, almacenamos la distancia desde el nodo raíz al actual (altura) y el número de arcos desde el actual al nodo hoja más distante (profundidad). Para posibilitar obtener información con respecto a los hijos de la sentencia se almacenan las categorías de sus tres primeros hijos.

A continuación, explicamos las características extraídas para cada construcción sintáctica.

### Programs

La Tabla 1 muestra la información almacenada para cada programa. Incluimos diferentes porcentajes definidos en ese programa (clases, enumerados, funciones…), todos ellos obtenidos de la sintaxis de sus hijos, e información sobre la implementación de las clases en paquetes.

Tabla 1: Características de programa.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Name | Nombre del proyecto | String |
| Has subdirs with code | Si en el directorio base del proyecto hay algún subdirectorio con ficheros Python pero sin un fichero \_\_init\_\_.py en el | True or False. |
| Has packages | Si en el directorio base del proyecto hay algún subdirectorio con ficheros Python y un fichero \_\_init\_\_.py en el | True or False |
| Number of modules | Número de ficheros Python en el total del proyecto | Integer |
| Number of subdirs with code | Número de subdirectorios con ficheros Python en su interior | Integer |
| Number of packages | Número de subdirectorios con ficheros Python en su interior, pero sin un fichero \_\_init\_\_.py | Integer |
| Class defs pct | Proporción de las definiciones dentro del proyecto que son definiciones de clases | [0, 1] |
| Function defs pct | Proporción de las definiciones dentro del proyecto que son definiciones de funciones | [0, 1] |
| Enum defs pct | Proporción de las definiciones dentro del proyecto que son clases enumeradas | [0, 1] |
| Has code root package | Si el proyecto tiene ficheros Python en el directorio base | True or False |
| Average defs per module | Número medio de definiciones que hay en cada fichero Python del proyecto | Real |
| Expertise level | Si el usuario que escribió este proyecto es experto o principiante | BEGINNER | EXPERT |
| UserID | Identificador del usuario que escribió este proyecto | Unique ID (Integer) |

### Modules

En lo que respecta a los módulos (ficheros Python), calculamos distintas características. Se incluyen el nombre del fichero y la convención de nombrado que sigue, también se incluyen diferentes proporciones de definiciones (clases, funciones y enumerados). Además, se incluye información relativa a las proporciones de sentencias y expresiones presentes en el fichero, contadores de funciones y clases y, alguna información extra extraída de los hijos del módulo. La Tabla 2 muestra las características almacenadas para un módulo.

Tabla 2: Características para el módulo.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Name | Nombre del fichero Python | String |
| Name convention | La convención utilizada para el nombre del fichero. | Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention |
| Has doc string | Si el fichero tiene un comentario de módulo. Esto es una cadena como primer hijo del módulo | True or False. |
| Global statements pct | Proporción de los hijos del módulo que son sentencias (sin contar imports y definciones) | [0, 1] |
| Global expressions pct | Proporción de los hijos del módulo que son expresiones | [0, 1] |
| Number of classes | Número de clases definidas en el módulo | Integer |
| Number of functions | Número de funciones definidas en el módulo | Integer |
| Class defs pct | Proporción de las definiciones de este módulo que son clases | [0, 1] |
| Function defs pct | Proporción de las definiciones de este módulo que son funciones | [0, 1] |
| Enum defs pct | Proporción de las definiciones de este módulo que son clases enumeradas | [0, 1] |
| Average statements function body | Número medio de sentencias en el cuerpo de las funciones de este módulo | Real |
| Average statements method body | Número medio de sentencias en el cuerpo de los métodos de este módulo | Real |
| Type annotation pct | Proporción de anotaciones de tipo en los parámetros y en las funciones | [0, 1] |
| Has entry point | Si el fichero tiene el idiom “if \_\_name\_\_ == ‘\_\_main\_\_’” | True or False |

### Imports

Esta información es complementaria a la tabla de módulos ya que almacena la información de los *imports* (una entrada por cada módulo).

Tabla 3: Características para los imports.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Number imports | Número de imports distintos en el módulo | Integer |
| Module imports pct | Proporción de imports simples (nodo Import) | [0, 1] |
| Average imported modules | Número medio de elementos importados por cada nodo Import | Real |
| From imports pct | Proporción de imports from (nodo ImportFrom) | [0, 1] |
| Average from imported modules | Número medio de elementos importados por cada nodo ImportFrom | Real |
| Average as in imported modules | Número medio de elementos importados con un alias con respecto a los nodos ImportFrom | Real |
| Local imports pct | Proporción de imports que no son definidos al comienzo del fichero | [0, 1] |

### Class Definitions

Respecto a la definición de clases, almacenamos la convención que sigue y la longitud del nombre de la clase. Contabilizamos además los decoradores de la clase, los métodos, las clases base que utiliza, el número de sentencias que forma la clase y el número de *keywords* (distintas de “meta=”) que utiliza la clase. Además, guardamos si la clase es un enumerado y si la clase tiene un comentario de clase (su primer hijo es una cadena), si la clase tiene una anotación de tipo genérica (por ejemplo, “class list[T]”) y si la clase usa una meta clase (define el *keyword* “meta=” en la cláusula de herencia).

Para identificar los tipos de métodos que se definen en la clase hemos identificado 7 tipos de métodos especiales: métodos privados, métodos mágicos, métodos asíncronos, métodos de clase, métodos estáticos, métodos abstractos y métodos de propiedad. Estos tipos se identifican en función de su nombre (por ejemplo, un método privado empieza por \_) o por las anotaciones que tiene el método (por ejemplo, @classmethod para un método de clase). De cada uno de estos tipos almacenamos la proporción de estos métodos que hay definidos en la clase.

Por último, guardamos las proporciones de anotaciones de tipos en los parámetros y tipos de retorno de los métodos de la clase, la proporción de expresiones que se definen en el cuerpo de la clase, proporción de asignaciones que se definen en el cuerpo de la clase, el número medio de sentencias en los cuerpos de los métodos y el código fuente de la propia clase como cadena de texto.

Tabla 4: Características para la definición de clases.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Name convention | La convención utilizada para el nombre de la clase. | Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention |
| Is enum class | Si la clase es una clase enumerada (hereda de la clase Enum) | True or False |
| Number of characters | Número de caracteres del nombre de la clase | Integer |
| Number of decorators | Número de decoradores de la clase | Integer |
| Number of methods | Número de métodos definidos en la clase | Integer |
| Number of base classes | Número de clases base que se definen en la clase | Integer |
| Has generic type annotations | Si la clase tiene una anotación de tipo genérico | True or False. |
| Has doc string | Si la clase tiene un comentario de clase. Esto es si su primer hijo es una cadena | True or False. |
| Body count | Número de sentencias en el cuerpo de la clase | Integer |
| Assignments pct | Proporción de las sentencias en el cuerpo de la clase que son asignaciones | [0, 1] |
| Expressions pct | Proporción de las sentencias en el cuerpo de la clase que son expresiones | [0, 1] |
| Uses metaclass | Si la clase usa una meta class. Esto es si define una keyword “meta=” en la cláusula de herencia | True or False. |
| Number of keywords | Número de keywords diferentes de metaclass que tiene la clase | Integer |
| Height | Distancia (número de nodos) desde la definición de la clase hasta el nodo del módulo en el que está | Integer |
| Average stmts method body | Número medio de sentencias en el cuerpo de los métodos | Real |
| Type annotations pct | Proporción de los métodos y de los parámetros de los métodos con anotación de tipos | [0, 1] |
| Private methods pct | Proporción de los métodos que son privados | [0, 1] |
| Magic methods pct | Proporción de los métodos que son magic | [0, 1] |
| Async methods pct | Proporción de los métodos que son asíncronos | [0, 1] |
| Class methods pct | Proporción de los métodos que son de clase | [0, 1] |
| Static methods pct | Proporción de los métodos que son estáticos | [0, 1] |
| Abstract methods pct | Proporción de los métodos que son abstractos | [0, 1] |
| Property methods pct | Proporción de los métodos que son de propiedad | [0, 1] |

### Function Definitions

Tabla 5: Características para definiciones de funciones.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Name convention | La convención utilizada para el nombre de la función. | Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention |
| Number of characters | Número de caracteres del nombre de la función | Integer |
| Is private | Si la función es privada o no | True or False |
| Is magic | Si la función es mágica o no | True or False |
| Is async | Si la función es asíncrona o no | True or False |
| Height | Distancia (número de nodos) desde la definición de función hasta la raíz del módulo | Integer |
| Body count | Número de sentencias en el cuerpo de la función | Integer |
| Expressions pct | Proporción de las sentencias en el cuerpo de la función que son expresiones | [0, 1] |
| Number of decorators | Número de decoradores que tiene la función | Integer |
| Has return type annotation | Si la función tiene anotación del tipo que devuelve | True or False |
| Has doc string | Si la función tiene un comentario de función. Esto es si su primer hijo es una cadena | True or False |
| Type annotations pct | Proporción de los parámetros que tiene anotación de tipo. Incluido el tipo de retorno | [0, 1] |

La Tabla 5 muestra la información almacenada para cada una de las definiciones de funciones. Guardamos la convención y el número de caracteres del nombre de la función. Muchas de las características que guardamos en esta tabla coinciden con las almacenadas para las definiciones de clases, por ejemplo, la proporción de expresiones en el cuerpo de la función, el número de decoradores, la proporción de anotaciones de tipos y el código fuente.

Para las funciones hemos identificado tres tipos: funciones privadas, funciones asíncronas y funciones mágicas.

### Method Definitions

Tabla 6: Características de definiciones de métodos.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Is class method | Si el método es de clase. Esto es si tiene el decorador @classmethod | True or False |
| Is static method | Si el método es estático. Esto es si tiene el decorador @staticmethod | True or False |
| Is constructor method | Si el método es un constructor. Esto es si se llama \_\_init\_\_ | True or False |
| Is abstract method | Si el método es abstracto. Esto es si tiene el decorador @abstract | True or False |
| Is property | Si el método es de propiedad. Esto es si tiene el decorador @property | True or False |
| Is cached | Si el método es cacheado. Esto es si tiene el decorador @cache | True or False |
| Is wrapper | Si el método es un *wrapper*. Esto es si tiene el decorador @wraps | True or False |

La Tabla 6 muestra la información adicional que complementa la tabla de definiciones de funciones para aquellas funciones que han sido definidas dentro de una clase (métodos). En esta tabla la única información que vamos a almacenar es la respectiva al tipo de método. Tendremos en cuenta los tipos de métodos ya descritos en el apartado de las definiciones de clases (estático, abstracto, de propiedad y de clase) y, añadiremos tres tipos más: constructor (si el nombre del método es \_\_init\_\_), cacheado (si tiene el decorador @cache) y *wrapper* (si tiene el decorador @wraps).

### Statements

En esta tabla vamos a agrupar todas las sentencias (teniendo un campo para distinguir los diferentes tipos) exceptuando las definiciones de funciones que hemos mostrado en los apartados anteriores y las expresiones que mostraremos en apartados posteriores.

Las sentencias van a tener tres campos de especial importancia: la categoría, que representa la categoría sintáctica de la sentencia; el rol de sentencia, que representa el rol que cumple la sentencia en su padre; y, la categoría del padre. Para estos tres campos definimos el dominio de posibles valores que pueden tomar (Anexo 9.2).

Además de esos tres campos, vamos a guardar su profundidad y su altura, las categorías sintácticas de sus tres primeros hijos y, solo para algunos tipos de sentencia (If, While, With, …), el número de sentencias de su cuerpo y si tienen una cláusula else.

Tabla 7: Características de sentencias.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Category | Categoría sintáctica de la sentencia | StatementCategory\* |
| Parent | Categoría sintáctica del nodo padre | Module | ClassDef | FunctionDef | MethodDef | StatementCategory\* |
| Statement role | Rol que cumple la sentencia en el nodo padre | StatementRole\* |
| Depth | Distancia máxima desde la sentencia hasta un nodo hoja | Integer |
| Height | Distancia desde el nodo actual hasta el nodo raíz. | Integer |
| Has or else | Si la sentencia tiene una cláusula else. Solo aplicable para Try, TryStar, If, For, AsyncFor y While. N/A en otro caso. | True, False, N/A |
| Body size | Número de sentencias en el cuerpo de la sentencia. Solo aplicable para While, If, For, AsyncFor, Try, TryStar, With y AsyncWith. N/A en otro caso. | True, False, N/A |
| First, second and third child | Categoría sintáctica del primer, segundo y tercer hijo de la sentencia respectivamente | Parameter| ExpressionCategory\* |

\* Dominio definido en el Anexo 9.1.

### Cases

Tabla 8: Características de los cases.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Number of cases | Número de cases en la sentencia Match | Integer |
| Guards | Proporción de *guards* en función del número de cases. | [0, 1] |
| Average body count | Número medio de sentencias en el cuerpo de los *cases* | Real |
| Average match value | Número medio de cláusulas MatchValue dentro de los *cases* de la sentencia Match | Real |
| Average match singleton | Número medio de cláusulas MatchSingleton dentro de los *cases* de la sentencia Match | Real |
| Average match sequence | Número medio de cláusulas MatchSequence dentro de los *cases* de la sentencia Match | Real |
| Average match mapping | Número medio de cláusulas MatchMapping dentro de los *cases* de la sentencia Match | Real |
| Average match class | Número medio de cláusulas MatchClass dentro de los *cases* de la sentencia Match | Real |
| Average match star | Número medio de cláusulas MatchStar dentro de los *cases* de la sentencia Match | Real |
| Average match as | Número medio de cláusulas MatchAs dentro de los *cases* de la sentencia Match | Real |
| Average match or | Número medio de cláusulas MatchOr dentro de los *cases* de la sentencia Match | Real |

En esta tabla almacenaremos información adicional a la tabla de sentencias para las sentencias Match. En este caso vamos a guardar el número medio de cada uno de los 8 tipos de cláusulas (MatchValue, MatchSingleton, MatchSequence, MatchMapping, MatchClass, MatchStar, MatchAs y MatchOr), además del número medio de sentencias en los cuerpos de las cláusulas *case*, el número de *cases* total y la proporción de *guards* en relación con total de *cases*.

### Handlers

Tabla 9: Características de los handlers.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Number of handlers | Número de cláusulas except | Integer |
| Has finally | Si la sentencia Try tiene una cláusula Finally | True or False |
| Has catch all | Si la sentencia Try tiene una cláusula except que capture todas las excepciones (type==None) | True or False |
| Average body count | Número medio de sentencias en el cuerpo de las cláusulas except. | Real |
| Has star | Si incluye una cláusula except con estrella (TryStar) | True or False |

En esta tabla vamos a almacenar información adicional a la tabla de sentencias para las sentencias Try y TryStar, concretamente en relación con el contenido y forma de las cláusulas except. Guardaremos: el número de *handlers* (cláusulas except); si el *try* contiene la cláusula finally; si el *try* contiene un *catch all* (except que admite cualquier tipo de excepción); el número medio de sentencias en el cuerpo de los *except*s y, si algún *handler* incluye el operador estrella (si la sentencia es un TryStar, contiene un except\*\*).

### Expressions

Tabla 10: Características de expression.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Category | Categoría sintáctica del nodo. | ExpressionCategory\* |
| First, second, third and fourth child | Categoría sintáctica del hijo correspondiente. | ExpressionCategory\* |
| Parent | Categoría sintáctica del nodo padre. | Module | ClassDef | FuncionDef | MethodDef | StatementCategory\* | ExpressionCategory\* |
| Expression role | Rol del nodo actual en el nodo padre. | ExpressionRole\* |
| Height | Distancia en arcos desde el nodo actual hasta el nodo raíz. | Integer |
| Depth | Distancia en arcos desde el nodo actual hasta el nodo hoja más distante. | Integer |

\* Dominio definido en el Anexo 9.1.

La Tabla 10 muestra la información almacenada para las expresiones. Almacenamos una información similar a la que almacenamos para las sentencias. En este caso, además de la categoría sintáctica de la expresión y de su nodo padre, almacenamos las categorías sintácticas de sus cuatro primeros hijos. También guardamos información sobre el papel que desempeña en su padre (rol), junto con los valores de la altura y la profundidad.

En los apartados siguientes vamos a comentar las diferentes tablas adicionales utilizadas para completar la información específica de algunas expresiones.

### Comprehensions

Tabla 11: Características de comprehensiones.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Category | Categoría sintáctica de la *comprehension* | ListComp | SetComp | DictComp | GenComp |
| Number of ifs | Número de condiciones de la *comprehension* (Ifs) | Integer |
| Number of generators | Número de generadores en la *comprehension* | Integer |
| Is async | Si la *comprehension* es asíncrona | True or False |

Esta es una tabla específica de expresiones, contiene la información relativa a las *comprehensions*. Esto son las expresiones de las siguientes categorías sintácticas: ListComp, SetComp, DictComp y GenComp. En esta tabla, a parte del tipo de *comprehension* que es, almacenaremos el número de generadores y el número de condiciones (Ifs) de la *comprehension*. Además, guardaremos si la *comprehension* es asíncrona.

### CallArgs

Tabla 12: Características de las invocaciones a funciones.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Number args | Número de argumentos en la invocación a la función | Integer |
| Named args pct | Proporción de argumentos pasados por referencia | [0, 1] |
| Double star args pct | Proporción de argumentos con la sintaxis \*\*args | [0, 1] |

Esta tabla almacena la información extra necesaria para las expresiones con categoría sintáctica Call (invocación a función). Esta información extra está formada por: número de argumentos; proporción de argumento parados por referencia (arg\_name=arg\_value) y, proporción de argumentos con la sintaxis “\*\*arg”.

### FStrings

La información relativa a las cadenas de texto formateadas (expresiones de la categoría JoinedStr) la guardaremos en esta tabla. Esta información será el número de elementos de la cadena formateada (constantes más valores formateados), la proporción de constantes en los elementos y la proporción de expresiones en los elementos.

Tabla 13: Características de las cadenas formateadas.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Number of elements | Número de elementos en la cadena formateada. Tanto constantes como valores formateados. | Integer |
| Constants pct | Proporción de los elementos anteriormente mencionados que son constantes. | [0, 1] |
| Expressions pct | Proporción de los elementos anteriormente mencionados que son expresiones (FormattedValues) | [0, 1] |

### Variables

En este caso, esta tabla contiene la información relativa a las variables. La información almacenada es: la convención de nombrado que sigue la variable; el número de caracteres del nombre; si la variable es privada, en función de su nombrado y, si la variable es mágica, en función también de su nombrado.

Tabla 14: Características de las variables.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Name convention | Convención de nombrado que sigue la variable. | Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention |
| Number of characters | Número de caracteres del nombre de la variable | Integer |
| Is private | Si la variable es privada. Esto es si su nombre comienza por \_ | True or False |
| Is magic | Si la variable es mágica. Esto es si su nombre es de la forma \_\_name\_\_ | True or False |

### Vectors

Esta es la última tabla del conjunto de tablas de expresiones. En esta recogeremos la información necesaria para los vectores. Los vectores son todas las expresiones que pertenezcan a una de las siguientes categorías sintácticas: ListLiteral, SetLiteral, DictLiteral y GeneratorLiteral. La información almacenada será: la categoría del vector, para diferenciar entre las 4 posibles; el número de elementos que componen el vector y, si el vector es homogéneo, esto es si todos los elementos que lo componen son del mismo tipo.

Tabla 15: Características de los vectores.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Category | Categoría sintáctica del vector | ListLiteral | SetLiteral | DictLiteral | GeneratorLiteral |
| Number of elements | Número de elementos que componen el vector | Integer |
| Homogeneous | Si todos los elementos del vector son del mismo tipo | True or False |

### Parameters

En esta tabla, almacenaremos la información relativa a los parámetros declarados tanto en la definición de funciones, métodos y constructores, como en las lambda expresiones. Para distinguir entre los parámetros que provienen de cada una de las posibilidades vamos a almacenar su rol. Además, vamos a almacenar la proporción de los parámetros que son de cada posible tipo: parámetros posicionales; parámetros variables; parámetros únicamente con *keyword* y parámetros con valor por defecto. Para algunos de estos tipos incluiremos un valor booleano para saber si tienen al menos un parámetro de ese tipo. Por último, incluiremos el convenio de nombrado más habitual entre los nombres de los parámetros y la proporción de parámetros con anotación de tipo.

Tabla 16: Características de los parámetros.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Parameters role | Indica el nodo en el que se definieron los parámetros | FunctionParameters | LambdaParameters |
| Number of params | Número de parámetros | Integer |
| Por only param pct | Proporción de parámetros posicionales. | [0, 1] |
| Var param pct | Proporción de parámetros como variable | [0, 1] |
| Has var param | Si hay algún parámetro del tipo variable | True or False |
| Type annotation pct | Proporción de los parámetros que tienen una anotación de tipo | [0, 1] |
| Kw only param pct | Proporción de los parámetros que solo se pueden pasar por referencia | [0, 1] |
| Default value pct | Proporción de los parámetros que definen un valor por defecto | [0, 1] |
| Has kw param | Si hay algún parámetro del tipo keyword | True or False |
| Name convention | Convención de nombrado más seguida por los nombres de los parámetros | Lower | Upper | CamelLow | CamelUp | SnakeCase | Discard | NoNameConvention |

Las 16 tablas creadas contienen información sobre construcciones sintácticas homogéneas, permitiéndonos obtener patrones comunes de definiciones, sentencias o expresiones. Sin embargo, un programa se compone de distintas construcciones, representado mediante información heterogénea (p. ej., un programa tiene un módulo, que tiene una clase, que define un método, donde se usa una sentencia que incluye a su vez varias expresiones distintas). Por ello, además de utilizar la información homogénea, debemos ofrecer la posibilidad de obtener información heterogénea y así poder trabajar con patrones sintácticos más expresivos y heterogéneos, formados por características de varias construcciones sintácticas. La creación de conjuntos de datos heterogéneos se puede realizar mediante operaciones de disgregación de datos (*drill down*) que combinan la información de dos o más tablas [5]. Así, conseguimos relacionar características de diferentes construcciones sintácticas como programas, definiciones, sentencias y expresiones. Como hemos comentado anteriormente, la tabla *nodes* contiene los identificadores de todos los nodos y sus correspondientes padres. De este modo, mediante la utilización de las sentencias SQL where y join es posible unir los datos de las diferentes tablas y obtener conjuntos de datos heterogéneos.

En resumen, nuestro sistema crea 16 conjuntos de datos homogéneos para el análisis y explotación de construcciones sintácticas, pero también ofrece la posibilidad de crear conjuntos de datos heterogéneos.

## Detección de valores atípicos

Los datos utilizados pueden contener instancias con valores atípicos, representando construcciones sintácticas significativamente distintas al resto de la población. La identificación y análisis de anomalías (*outliers*) es muy importante para ofrecer una información valiosa y precisa del conjunto de datos obtenido en este trabajo. Adicionalmente, determinadas anomalías pueden deberse a errores en los datos, que deben ser eliminadas para evitar conclusiones erróneas [6].

Nuestros conjuntos de datos poseen información numérica y categórica. Para detectar valores anómalos univariable en los datos numéricos, empleamos el test de Tukey basado en comparaciones con el rango intercuartil (IQR), definido como la diferencia entre el tercer y primer cuartil de una distribución (IQR = Q3 – Q1), donde Qn representa el cuartil n [7].

De este modo, se define un valor atípico como aquél fuera de alguno de los dos siguientes intervalos:

(1)

(2)

El primer rango permite identificar un valor atípico leve y el segundo uno extremo. En base a la distribución de los valores de la variable, consideraremos (1) o (2) como rangos para la detección de anomalías, eligiendo (2) siempre que haya valores atípicos extremos y (1) en caso contrario.

Los límites de Tukey se basan en los cuartiles de los datos y son sensibles a la presencia de sesgo en la distribución. Cuando hay asimetría en los datos, los límites de Tukey pueden no ser tan efectivos para identificar *outliers* de manera equitativa en ambos extremos de la distribución. Los métodos *Adjusted Box Plots* están diseñados para describir distribuciones sesgadas y se basan en medidas de asimetría. El Coeficiente de Medcouple (*Medcouple Coefficient*, *MC*) es útil para identificar la asimetría en los datos, especialmente en presencia de valores atípicos o sesgados. Es una medida robusta porque no se ve tan afectada por valores extremos como la media y la desviación típica. El *MC* puede proporcionar información adicional sobre la asimetría de la distribución, lo que te permite ajustar los límites de Tukey de manera más apropiada para una distribución de datos específica. En [8] proponen el siguiente método para ajustar los límites de Tukey en función del *MC*:

(3)

(4)

En este caso, utilizamos diferentes rangos según la dirección de la asimetría en los datos. Si el coeficiente de asimetría *MC* es mayor que 0, lo que indica una asimetría hacia la derecha, utilizamos el primer rango (3) para identificar valores atípicos. Si la asimetría es hacia la izquierda (MC menor que 0), empleamos el segundo rango (4). Cuando aplicamos el test de Tukey y detectamos valores atípicos extremos, utilizamos los rangos (3) o (4), según la dirección de la asimetría indicada por el MC, para verificar si el número de valores atípicos obtenidos es menor. Si este es el caso, dicho rango será el utilizado para considerar los valores como anómalos.

Para la detección de anomalías multivariable, se utilizó el algoritmo de *Isolation Forest*. Este algoritmo identifica los valores atípicos teniendo en cuenta lo lejos que está un punto de datos (instancia) del resto de los datos [9]. El hiperparámetro de contaminación especifica la proporción de valores atípicos en el conjunto de datos. En nuestro caso encontramos que el valor 0,01 (1%) es el factor de contaminación que mejor identifica los valores atípicos en nuestros conjuntos de datos.

Para el caso de las variables categóricas, empleamos un análisis de frecuencia en el que consideramos que un valor categórico es atípico cuando su número de ocurrencias es menor que 0,2% / *número posibles valores*. Por ejemplo, una variable booleana tendrá un valor anómalo cuando éste ocurra menos del 0,1% del total.

# Metodología

## Conjunto de Datos

El conjunto de datos utilizado incluye programas escritos tanto por programadores principiantes como por expertos (Tabla 17). Los programas de principiantes han sido obtenidos de estudiantes de primer año del Grado en Ingeniería de Software de la Universidad de Oviedo, durante el periodo 2020-2024, de la asignatura de Fundamentos de la Informática. Cada programa representa un examen o una práctica realizada por un alumno o por un grupo de alumnos. Para poder identificar los programas, hemos tenido en cuenta la estructura de directorios y la presencia de ficheros Python en dichos directorios.

El código fuente de los programas escritos por expertos fue obtenido utilizando la *API* de *GitHub*, la cual permite obtener repositorios públicos con licencia *open source*. Aunque los detalles exactos del algoritmo utilizado por GitHub no están públicamente documentados, se sabe que la *API* tiene en cuenta factores como la fecha de última actividad, la relevancia del título y la descripción, así como el número de estrellas, contribuidores y *forks* para determinar la relevancia de los repositorios devueltos. Dado que en este proyecto nos enfocamos exclusivamente en proyectos escritos en Python, utilizamos la *API* de *GitHub* para filtrar y obtener únicamente repositorios que estuvieran etiquetados como proyectos escritos en este lenguaje de programación. Una vez obtenida la lista de resultados (en el Anexo 9.3 está disponible la lista completa), utilizamos *Git* para clonar estos repositorios y obtener así el código fuente completo. Este proceso nos ha permitido analizar grandes volúmenes de código Python provenientes de proyectos reales y diversos.

Finalmente, el conjunto de datos utilizado en este trabajo está formado por 1.609 programas con 18.226 ficheros Python.

Tabla 17: Número de Nodos de los ASTs

|  |  |  |  |
| --- | --- | --- | --- |
|  | Principiante | Experto | Total |
| PROGRAMAS | 1.591 | 18 | 1.609 |
| Modules | 7.124 | 11.102 | 18.226 |
| Imports | 7.124 | 11.102 | 18.226 |
| Classdefs | 7.755 | 14.569 | 22.324 |
| Functiondefs | 59.879 | 88.089 | 147.968 |
| Methoddefs | 35.666 | 56.754 | 92.420 |
| Parameters | 61.752 | 92.146 | 153.898 |
| Expressions | 3.490.521 | 4.792.486 | 8.283.007 |
| Callargs | 338.727 | 521.668 | 860.395 |
| Comprehensions | 4.982 | 13.277 | 18.259 |
| FStrings | 4.977 | 17.053 | 22.030 |
| Variables | 1.167.967 | 1.694.338 | 2.862.305 |
| Vectors | 121.764 | 229.900 | 351.664 |
| Statements | 457.483 | 617.277 | 1.074.760 |
| Cases | 0 | 29 | 29 |
| Handlers | 8.953 | 6.803 | 15.756 |
| Total |  |  | **13.942.876** |

En la Tabla 17, podemos observar que la distribución de los datos para cada clase de programador se encuentra equilibrada, excepto en el caso de los programas. Esto se debe a que los proyectos expertos son de mayor entidad y contienen un mayor número de archivos. En la construcción del conjunto de datos, se buscaba lograr un mayor equilibrio en las demás construcciones sintácticas, independientemente del número de programas.

A partir del código fuente, se generan los *AST*s y se recorren éstos para para rellenar las 16 tablas de definidos en las Sección 3.2. La Tabla 17 muestra el número de nodos *AST* obtenidos para cada una de las tablas, observándose cómo la base de datos tiene un total de casi 14 millones de entradas.

## Entorno de ejecución

Nuestro sistema se ha implementado modificando el compilador de Java del OpenJDK 11.0.3, y desarrollando el resto de los módulos en Python 3.7.4 con scikit-learn 0.21.3 y mlxtend 0.18.0. Para almacenar los *datasets* utilizamos PostgreSQL 10.15. Todo el código fue ejecutado en un servidor Dell PowerEdge R530 con dos procesadores Intel Xeon E5-2620 v4 2.1 GHz (32 núcleos) con 128GB DDR4 2400 MHz de memoria RAM, ejecutando un sistema operativo CentOS 7.9–2009.1 de 64 bits.

# Evaluación

Nuestro conjunto de datos consta de un total de 13.942.876 nodos extraídos de los *AST*s generados por el conjunto de programas seleccionado. Todos estos nodos pertenecen a una de las 17 tablas definidas (en la Tabla 17 se puede ver el desglose de los datos).

Siguiendo el método descrito en la Sección 3.3., hemos realizado el análisis de valores anómalos. A continuación, se detallan los resultados más destacables. El informe completo del análisis realizado se puede consultar en el Anexo 9.2.

## Programas

1. Se detectaron un total de 544 programas, cerca de un 34%, que no tienen definiciones (clases, funciones o enumerados).
2. Un 98% de los programas no contienen ningún paquete. Todos los módulos están almacenados en la raíz del programa.
3. Se detectaron dos programas, ambos de expertos, que tienen más de 3.000 módulos, mientras que la media es de 11,3 módulos por programa. Estos programas son ray y llama\_index, ambos con más de 900 contributors. Además, de entre los programas de alumnos, hay 6 que tiene más de 500 módulos. Esto se debe a la entrega de librerías Python completas por lo que se han descartado.
4. Se ha detectado un programa con más de 200 subdirectorios con código, mientras que la media es de 0,4 subdirectorios por programa. Este programa es ray, que como se comento anteriormente, cuenta con más de 900 contributors y es un proyecto enorme.
5. Se ha detectado un programa con más de 1.000 paquetes, mientras que la media es de 1,64 paquetes por programa. Además, se ha detectado otro programa con cerca de 400 paquetes, mientras que el siguiente programa con más paquetes tiene no más de 150. Estos dos programas son ray y llama\_index. En el caso de los programas de principiantes, en relación con lo indicado en el punto 3, hay varios programas con un número de paquetes muy superior al resto.
6. Se detectaron como anómalos los programas que tienen una o más enumeraciones, debido a que tan solo el 0,7% de los programas tienen alguna enumeración. Además, todos los programas que tienen alguna enumeración fueron escritos por expertos.
7. De entre los programas de principiantes no hay ninguno que no tenga ficheros con código en el directorio raíz del programa. Esto se debe a la estructura sencilla de las entregas que, en la mayoría de los casos, era una estructura obligatoria.
8. Mientras que en los programas de expertos no hay ningún programa con ninguna definición de clases, entre los programas de principiantes, solo un 2% tiene al menos una clase definida.

## Módulos

1. Se detectaron un total de 3.974 módulos que no tenían definiciones (clases, funciones o enumerados), de los cuales, 2.199 son de expertos.
2. Se han detectado 3 módulos con un número de clases superior a 90, mientras que la media es de 1 clase por módulo. Todos ellos, han sido escritos por expertos.
3. Se ha detectado 1 módulo con más de 185 funciones definidas, mientras que la media es de 2,5 definiciones de funciones por módulo.
4. Se ha detectado un módulo con más de 100 sentencias de media en los cuerpos de sus métodos. Este valor se encuentra muy alejado del resto, siendo la media de 1,66. Así mismo, hay otros 2 módulos con más de 35 sentencias de media, siendo todos ellos escritos por expertos, y siendo muy superiores a la media.
5. Se ha detectado que el 56,5% de los módulos no tienen expresiones declaradas en el ámbito global del módulo. De este 56,5%, el 65% fueron escritos por expertos.
6. Entre los expertos, el convenio de nombrado de módulos más habitual es el SnakeCase, con cerca de un 65% de uso. Por el contrario, entre los principiantes el SnakeCase solo se usa un 20% de los casos, y es el Lower el más usado en un 45% de los módulos.
7. Entre los expertos, cerca del 50% de los módulos no tienen ninguna definición de clase, entre los principiantes, este porcentaje alcanza el 70%. [esto no contradice el punto 9 anterior??] – una cosa es programas y otra módulos.
8. Entre los principiantes, cerca del 50% de los módulos solo tienen definiciones de funciones. Por el contrario, entre los expertos, un 50% de los módulos no tienen definiciones de funciones y tan solo el 30% están formados únicamente por definiciones de funciones.
9. Entre los expertos, un 41% de los módulos tienen al menos una anotación de tipo, este porcentaje entre los principiantes es de tan solo el 12%. Entre los principiantes no existe ningún módulo con un 100% de anotaciones de tipos.

## Imports

1. Se ha detectado que los imports del tipo as in solo se usan en un 6% de los módulos.
2. Se ha detectado que tan solo el 18% de los módulos no tienen ningún import.
3. Se han detectado 3 ficheros con un número de imports anómalo, se detectaron anómalos al tener más de 123 imports, mientras que la media es de 5,11.
4. Se ha detectado que un 40% de los módulos tienen 0 imports simples. En los principiantes, este porcentaje se eleva hasta un 52%, mientras que en expertos este porcentaje es del 37%.
5. Se ha detectado que un 28% de los módulos tienen 0 imports from, mientras que un 25% solo tienen imports de tipo from. En el caso de los principiantes, el 52% no tienen ningún import from y el 15% solo tienen imports de tipo from. Para los expertos, solo el 13% no tienen imports from, y el 31% solo tienen imports from.

## Definiciones de clase

1. Se ha detectado que tan solo el 13% de las definiciones de clase tienen decoradores. Para los principiantes este porcentaje se reduce hasta un 0,1%, y para los expertos aumenta hasta un 20%.
2. Se ha detectado que ninguna definición de clase tiene una anotación de tipo genérica. Esto puede deberse a que esa funcionalidad fue añadida a Python en la versión 3.12, la más reciente a la realización de este trabajo, por lo que todavía no habrán implementado esta característica del lenguaje.
3. Se ha detectado que tan solo un 1,3% de las clases son enumerados, todas ellas fueron escritas por expertos.
4. Se ha detectado como anómalo una clase con más de 200 métodos, mientras que la media es de 4,14 métodos por clase.
5. Se han detectado un total de 4000 clases sin métodos. Estas clases representan el 20% del total. Las clases sin métodos constan, en su mayoría, de un comentario de clase y una sentencia Pass. Todo apunta a que estas clases son fruto de herencia mal usada.
6. Se ha detectado como anómala cualquier clase que tenga más de una clase base. Esto se debe a que tan solo el 4,2% tienen más de una clases base. De entre las clases con más de una clase base, el 70% son de expertos y, existe una clase con un total de 45 clases base.
7. Se ha detectado una clase con una media de 114 sentencias en los cuerpos de sus métodos. Este valor está muy alejado del siguiente, la media de este valor es de 2,9.
8. Se ha detectado una clase que tiene un total de 618 sentencias en el ámbito global de la clase. Se aleja mucho del siguiente valor, la media de este valor es de 5,76.
9. Se ha detectado que el 69% de las clases no tienen ninguna sentencia en el ámbito global. Además, un 51% de las clases no tienen expresiones en el ámbito global.
10. Se ha detectado que el 70% de las clases no tienen ninguna anotación de tipo. En el caso de los principiantes se eleva hasta un 83% y, en el caso de los expertos, disminuye hasta el 63%.
11. Se ha detectado como anómalo la presencia de cualquier tipo de método especial, exceptuando los métodos mágicos. Esto se debe a que muy pocas clases presentan alguno de estos métodos. En el caso de los métodos privados, son un 19,05% las clases que tienen al menos un método; para los métodos asíncronos, son un 5,2%; para los métodos de clase, son un 8,32%; para los métodos estáticos, son un 4,4% y, para los métodos abstractos, son un 1,26%; para los métodos de propiedad, son un 9,56%.
12. Se ha detectado que un 88,4% de las clases tienen el convenio de nombrado CamelUp.

## Definiciones de funciones

1. Se ha detectado que tan solo el 15% de las funciones tienen al menos un decorador. Este porcentaje baja hasta el 2% en principiantes y aumenta hasta un 25% en las funciones de expertos.
2. Se ha detectado que el 73% de las funciones no tienen ninguna anotación de tipo. Este porcentaje aumenta hasta el 84% en principiantes y disminuye hasta un 64% en las funciones de expertos.
3. Se ha detectado que el 100% de las funciones con un número de caracteres anómalo han sido escritos por expertos. Además, el 99% de estas funciones tienen SnakeCase como convenio de nombrado.
4. Se ha detectado una función con un total de 172 sentencias en el ámbito global de la función. Este valor esta muy alejado del siguiente, la media es de 4,66 sentencias por función.
5. Se ha detectado una función con un total de 45 decoradores. Este valor esta muy alejado del siguiente, la media es de 0,18 decoradores por función. Además, de entre las funciones que tienen al menos un decorador, un 15%, el 94% son de expertos.
6. Se ha detectado que, con respecto a la altura de las funciones, el 83% tienen altura 1 (funciones en el ámbito global del módulo), y tan solo un 7% tienen una altura superior a 2.
7. Se ha detectado que un 42% de las funciones no tienen ninguna expresión en el ámbito global y casi un 4% tiene únicamente expresiones.
8. Se ha detectado que un 73% de las funciones no tienen ninguna anotación de tipo y menos de un 5% tienen la totalidad de anotaciones de tipos posibles. En principiantes es un 84% el que no tiene ninguna anotación de tipo y un 15% el que tiene la totalidad de anotaciones de tipos posibles. En expertos estos porcentajes son del 64% y del 27% respectivamente.

## Definiciones de métodos

1. No se ha detectado ningún método wrapper ni cached, ni entre los principiantes, ni entre los expertos.
2. Se ha detectado un método con 364 sentencias en el ámbito global. Es muy superior al siguiente valor, la media para esta variable es de 3,74 sentencias.
3. Se ha detectado que el 49% de los métodos no tienen ninguna expresión en el ámbito global. En los principiantes este porcentaje es del 51%, en los expertos es del 46%.
4. Se ha detectado que el 65% de los métodos no tienen anotaciones de tipo. En los principiantes este porcentaje se eleva hasta el 79%, en los expertos se reduce hasta el 54%.
5. Se ha detectado la convención de nombrado Discard como valor anómalo. En el caso de los principiantes ningún método tiene esta convención de nombrado, en los expertos también se detecta como anómalo.

## Sentencias

1. Se han detectado 6 sentencias con valores de profundidad superior a 80, mientras que la media es de 2,85.
2. Se ha detectado una sentencia con un número de sentencias o expresiones en el cuerpo de 276, muy superior al resto de valores. La media para esta variable es de 1,89.
3. En el caso de los principiantes, no se han detectado ninguna sentencia de los tipos Nonlocal, TypeAlias, ExceptHandler, Match, AsyncWith y AsyncFor. Para los expertos no se han detectado son ExceptHandler y TypeAlias.
4. Entre las sentencias de expertos, no existe ninguna con un padre del tipo AssignmentStmt, Import, Nonlocal, TypeAlias, Raise, Continue, Break, AsyncFor, ImportFrom, Assert, Return, Pass, AugmentedAssignment, AnnotatedAssignment, Delele y Global. Los principiantes, además de los padres no usados por los expertos, tampoco han usado ni Match ni AsyncWith.
5. En el caso de las sentencias usadas por expertos, ninguna cumple el rol de AsyncForElseBody, TryHandlerStar o TryHandler. Además de los roles no usados por expertos, los principiantes tampoco tienen sentencias que cumplan el rol de AsyncForBody, CaseBody o AsyncWithBody.
6. De entre las sentencias escritas por principiantes, ninguna sentencia tiene un primer hijo del tipo Slice, SetLiteral, FormattedValue, MatMult, Yield, EllipsisLiteral, Parameter, Star, UnaryBWNot, YieldFrom, NoneType o AssignmentExp. Las sentencias de principiantes, por su parte, nunca tienen un primer hijo del tipo FormattedValue, Star, YieldFrom, EllipsisLiteral, Yield, Parameter o Slice.
7. De entre las sentencias escritas por principiantes, ninguna sentencia tiene un segundo hijo del tipo YieldFrom, AssignmentExp, MatMult, Star, Parameter, EllipsisLiteral, FormattedValue y Slice. Los expertos nunca usan ningún segundo hijo del tipo AssignmentExp, FormattedValue, Star, Parameter y Slice.
8. De entre las sentencias escritas por expertos, ninguna tiene un tercer hijo del tipo UnaryBWNot, YieldFrom, Shift, AssignmentExp, MatMult, Star, Parameter, Pow, Yield, SetComprehension, ComplexLiteral, FString, UnaryNot, EllipsisLiteral, Compare, FormatterValue, Await y Slice. En el caso de los principiantes, los valores que nunca toma el tercer hijo son YieldFrom, EllipsisLiteral, Star, Yield, FormattedValue, UnaryBWNot, Slice, AssignmentExp, Parameter, MatMult, GeneratorComprehension, BWLogical y Shift.

## Cases

1. No se ha detectado entre los principiantes ninguna sentencia del tipo Match, por lo que no hay ninguna entrada de esta tabla escritas por principiantes.
2. Se han detectado dos Match con un número de cases superior a 20, mientras que la media es de 5,17 cases por cada Match.
3. No se ha detectado ningún case con un guard.
4. No se ha detectado ningún case de los tipos Singleton, Sequence, Or, Mapping, Class o Star.

## Handler

1. No se ha detectado ningún Try del tipo TryStar.
2. Se ha detectado un Try que solo cuenta con una cláusula finally sin ningún except, esto hace que tenga una media de sentencias en cada except de 0.

## Expresiones

1. Entre los expertos, no se ha detectado ninguna expresión de la categoría sintáctica NoneType. Por otro lado, los principiantes, además de no usar NoneType, no usaron MatMult.
2. Analizando las expresiones escritas por principiantes se ha detectado que no hay expresiones cuyo padre sea de los tipos NoneLiteral, Import, Global, FloatLiteral, NoneType, EllipsisLiteral, TypeAlias, Match, Pass, Continue, Nonlocal, ComplexLiteral, ImportFrom, MatMult, Variable, Break, StringLiteral, BoolLiteral, IntLiteral, AsyncFor y AsyncWith. Los expertos, nunca usan como padre ComplexLiteral, Global, ImportFrom, Continue, StringLiteral, NoneLiteral, Nonlocal, FloatLiteral, EllipsisLiteral, Import, NoneType, Break, IntLiteral, Pass, TypeAlias, BoolLiteral o Variable.
3. Se ha detectado que los expertos no han utilizado como primer hijo de una expresión otra de los tipos YieldFrom, Parameter, Yield y AssignmentExp. Por su parte, los principiantes no han usado Parameter, DictComprehension, Await, Yield, YieldFrom, Slice, Lambda y MatMult.
4. Como segundo hijo, los expertos nunca usan expresiones de los tipos MatMult, YieldFrom, NoneTyp, Parameter y Yield. Los principiantes, las categorías sintácticas que no usan como segundo hijo son Parameter, Await, Yield, YieldFrom, NoneType y MatMult.
5. De entre todas las expresiones de principiantes ninguna usa como tercer hijo Parameter, Await, Shift, Yield, UnaryBWNot, YieldFrom, EllipsisLiteral, Slice, SetComprehension y MatMult. Los expertos, las categorías que no usan son YieldFrom, Parameter, Yield y MatMult.
6. Entre los expertos, nunca se usa como cuarto hijo las categorías sintácticas MatMult, GeneratorComprehension, YieldFrom, NoneType, Parameter, UnaryBWNot, Yield y AssignmentExp. En las expresiones de principiantes, las categorías no usadas son Parameter, Await, Shift, Yield, UnaryBWNot, YieldFrom, EllipsisLiteral, Slice, Pow, AssignmentExp, NoneType, SetComprehension y MatMult.
7. Por último, los roles que nunca cumplen las expresiones escritas por expertos son TypeVar, MatMult, BWLogical, CaseGuard, AugmentedAssignmentLHS, TryElse, TypeAnnotation, TypeAliasRHS, AsyncForElseBody, Is, TypeAliasLHS, Pow, DefaultParamValue, AugmentedAssignmentRHS, ComprehensionElement, In y Shift. Los principiantes, los roles que nunca usan son AsyncWithElement, AugmentedAssignmentLHS, Shift, DefaultParamValue, MatchCondition, AsyncForBody, BWLogical, TryElse, In, Pow, TypeVar, AsyncWithAs, CaseBody, MatMult, CaseCondition, ComprehensionElement, AugmentedAssignmentRHS, TypeAnnotation, AsyncForEnumerable, CaseGuard, AsyncForElseBody, TypeAliasRHS, TypeAliasLHS, AyncWithBody, Is y AsyncForElement.

## Comprehensions

1. Se han detectado como anómalas las comprehensions asíncronas. En el caso de los principiantes, no se ha detectado ninguna de este tipo.

## Invocaciones a funciones

1. Se ha detectado una invocación a función con un número de argumento, 96, muy superior a los siguientes que tienen menos de 50.
2. Se ha detectado que el 87% de las invocaciones a funciones no tienen ningún argumento pasado por nombre. Este porcentaje baja al 82% en los expertos y se eleva hasta el 95% en principiantes.
3. Se ha detectado que a penas el 0,1% de las invocaciones a funciones tienen algún parámetro pasado por double star.

## Cadenas formateadas

1. Se han detectado varias cadenas formateadas cuyo número de elementos es muy superior a la media. En el caso de los principiantes hay una cadena formateada con 43 elementos cuando la media es 3,2, mientras que en los expertos hay tres cadenas con alrededor de 40 elementos cuando la media es de 3,16.

## Variables

1. Nada que destacar

## Vectores

1. Se ha detectado un vector con poco más de 14500 elementos, escrito por un experto, mientras que la media es de 4,37 elementos en cada vector.

## Parámetros

1. Se ha detectado que de todos los parámetros que declaran algún parámetro con keyword, el 82% pertenecen a expertos.
2. Se ha detectado que tan solo un 0,0052% de los parámetros tienen al menos un parámetro por posición. De este pequeño porcentaje, el 100% es de expertos.
3. Se ha detectado que de entre el 22% de los parámetros que tienen alguna anotación de tipo, el 80% son de expertos.

# Conclusiones y Trabajo Futuro

## Conclusiones

TODO

## Trabajo Futuro

Como trabajo futuro planeamos utilizar los *datasets* generados para TODO

Todos los datos utilizados en nuestro trabajo, el informe de valores anómalos, el nuevo diseño del AST y todo el código fuente utilizado para implementar nuestro sistema están disponibles para su descarga en <https://github.com/ComputationalReflection/PythonSourceCodeAnalysis>.

# Planificación y Presupuesto

## Planificación del proyecto

En la siguiente tabla se representan las fechas exactas y las horas de trabajo del proyecto. Las horas de trabajo diarias han variado en función de la carga de trabajo, la dificultad de este y la previsión de plazos restantes. Durante las fases 1, 3 y 4 hemos trabajado 2 horas diarias. En la fase 2 han sido 3 horas. Por último, en la fase de documentación, 4 horas diarias.

Tabla 18: Planificación del Proyecto

|  |  |  |  |
| --- | --- | --- | --- |
| **Tarea** | **Duración (h)** | **Comienzo** | **Fin** |
| ***Fase 1 (Diseño y análisis de la solución)*** | ***30*** | ***15/10/2023*** | ***02/11/2023*** |
| Decisión del alcance | 10 | 15/10/2023 | 19/10/2023 |
| Elección de tecnologías | 10 | 20/10/2023 | 26/10/2023 |
| Diseño de la arquitectura | 10 | 27/10/2023 | 02/11/2023 |
| ***Fase 2 (Implementación de la solución)*** | ***294*** | ***03/11/2023*** | ***19/03/2024*** |
| Desarrollo del programa | 150 | 03/11/2023 | 11/01/2024 |
| Desarrollo de la interfaz a la BD | 39 | 12/01/2024 | 30/01/2024 |
| Creación y configuración del entorno | 9 | 31/01/2024 | 02/02/2024 |
| Integración de las partes | 21 | 05/02/2024 | 13/02/2024 |
| Testing y corrección de errores | 75 | 14/02/2024 | 19/03/2024 |
| ***Fase 3 (Generación del dataset)*** | ***70*** | ***20/03/2024*** | ***07/05/2024*** |
| Obtención de programas | 10 | 20/03/2024 | 26/03/2024 |
| Ejecución del programa | 10 | 27/03/2024 | 02/04/2024 |
| Testing y corrección de errores | 50 | 03/04/2024 | 07/05/2024 |
| ***Fase 4 (Análisis del dataset)*** | ***60*** | ***08/05/2024*** | ***18/06/2024*** |
| Análisis del dataset | 30 | 08/05/2024 | 28/05/2024 |
| Testing y corrección de errores | 30 | 29/05/2024 | 18/06/2024 |
| ***Documentación del proyecto*** | ***60*** | ***19/06/2024*** | ***09/07/2024*** |
| ***Total*** | *514* | *15/10/2023* | *09/07/2024* |

## Planificación del proyecto

### Precios por hora

Todas las tareas completadas en las diferentes fases del proyecto han sido realizadas de forma secuencial. Una misma persona, tomando el rol de investigador y programador alternativamente ha desarrollado el proyecto en su conjunto.

En las tablas que siguen se representan los salarios por hora que hemos tenido en cuenta para cada uno de los roles. Estos salarios se han calculado teniendo en cuenta la situación actual del mercado laboral en España. En estos salarios hemos tenido en cuenta únicamente el coste directo, los costes indirectos del trabajo se pueden visualizar en el apartado Presupuesto Total.

Tabla 19: Precio por hora investigador

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Unidad** | **Descripción** | **Precio/hora (€)** | **Horas** | **Subtotal (€)** |
| Hora | Investigador | 50,00 € | 1 | 50,00 € |
|  |  |  | **Precio total / hora** | 50,00 € |

Tabla 20: Precio por hora programador

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Unidad** | **Descripción** | **Precio/hora (€)** | **Horas** | **Subtotal (€)** |
| Hora | Programador | 40,00 € | 1 | 40,00 € |
|  |  |  | **Precio total / hora** | 40,00 € |

### Precio por unidad de trabajo

El precio asociado a cada una de las tareas que componen el proyecto se listan a continuación, teniendo en cuenta el rol que ha realizado dicha tarea y el número de horas que ha llevado su realización.

Como se puede observar, el rol de programador esta asociado a las tareas que incluyen de forma directa la implementación de código, el manejo de sistemas informáticos y el testing. Por otro lado, el investigador se encarga del resto de tareas que, en resumen, son las relacionadas con los aspectos más teóricos y de decisión del proyecto.

Tabla 21: Precios por unidad de trabajo. Parte 1: Decisión de alcance

|  |  |  |  |
| --- | --- | --- | --- |
| **Decisión del alcance** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 10 | Investigador | 50,00 € | 500,00 € |
|  |  | **Precio total** | 500,00 € |

Tabla 22: Precios por unidad de trabajo. Parte 2: Elección de tecnologías

|  |  |  |  |
| --- | --- | --- | --- |
| **Elección de tecnologías** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 10 | Investigador | 50,00 € | 500,00 € |
|  |  | **Precio total** | 500,00 € |

Tabla 23: Precios por unidad de trabajo. Parte 3: Diseño de la arquitectura

|  |  |  |  |
| --- | --- | --- | --- |
| **Diseño de la arquitectura** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 10 | Investigador | 50,00 € | 500,00 € |
|  |  | **Precio total** | 500,00 € |

Tabla 24: Precios por unidad de trabajo. Parte 4: Desarrollo del programa

|  |  |  |  |
| --- | --- | --- | --- |
| **Desarrollo del programa** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 150 | Programador | 40,00 € | 6.000,00 € |
|  |  | **Precio total** | 6.000,00 € |

Tabla 25: Precios por unidad de trabajo. Parte 5: Desarrollo de la interfaz a la BD

|  |  |  |  |
| --- | --- | --- | --- |
| **Desarrollo de la interfaz a la BD** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 39 | Programador | 40,00 € | 1.560,00 € |
|  |  | **Precio total** | 1.560,00 € |

Tabla 26: Precios por unidad de trabajo. Parte 6: Creación y configuración del entorno

|  |  |  |  |
| --- | --- | --- | --- |
| **Creación y configuración del entorno** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 9 | Programador | 40,00 € | 360,00 € |
|  |  | **Precio total** | 360,00 € |

Tabla 27: Precios por unidad de trabajo. Parte 7: Integración de las partes

|  |  |  |  |
| --- | --- | --- | --- |
| **Integración de las partes** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 21 | Programador | 40,00 € | 840,00 € |
|  |  | **Precio total** | 840,00 € |

Tabla 28: Precios por unidad de trabajo. Parte 8: Testing y corección de errores

|  |  |  |  |
| --- | --- | --- | --- |
| **Testing y corrección de errores** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 75 | Programador | 40,00 € | 3.000,00 € |
|  |  | **Precio total** | 3.000,00 € |

Tabla 29: Precios por unidad de trabajo. Parte 9: Obtención de programas

|  |  |  |  |
| --- | --- | --- | --- |
| **Obtención de programas** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 10 | Investigador | 50,00 € | 500,00 € |
|  |  | **Precio total** | 500,00 € |

Tabla 30: Precios por unidad de trabajo. Parte 10: Ejecución del programa

|  |  |  |  |
| --- | --- | --- | --- |
| **Ejecución del programa** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 10 | Investigador | 50,00 € | 500,00 € |
|  |  | **Precio total** | 500,00 € |

Tabla 31: Precios por unidad de trabajo. Parte 11: Testing y corrección de errores

|  |  |  |  |
| --- | --- | --- | --- |
| **Testing y corrección de errores** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 50 | Programador | 40,00 € | 2.000,00 € |
|  |  | **Precio total** | 2.000,00 € |

Tabla 32: Precios por unidad de trabajo. Parte 12: Análisis del dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Análisis del dataset** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 30 | Investigador | 50,00 € | 1.500,00 € |
|  |  | **Precio total** | 1.500,00 € |

Tabla 33: Precios por unidad de trabajo. Parte 13: Testing y corrección de errores

|  |  |  |  |
| --- | --- | --- | --- |
| **Testing y corrección de errores** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 30 | Programador | 40,00 € | 1.200,00 € |
|  |  | **Precio total** | 1.200,00 € |

Tabla 3425: Precios por unidad de trabajo. Parte 14: Documentación del proyecto

|  |  |  |  |
| --- | --- | --- | --- |
| **Documentación del proyecto** | | | |
|  |  |  |  |
| **Cantidad (horas)** | **Descripción** | **Precio (€)** | **Subtotal (€)** |
| 60 | Investigador | 50,00 € | 3.000,00 € |
|  |  | **Precio total** | 3.000,00 € |

### Presupuesto total

La Tabla 20 muestra el presupuesto final del proyecto. En este presupuesto se incluyen los precios de cada una de las tareas listadas en el apartado anterior, además de los costes indirectos (un 10% de los costes directos). El listado de conceptos incluidos dentro de los costes indirectos puede verse listados en la Tabla 21.

Tabla 35: Presupuesto total

|  |  |  |
| --- | --- | --- |
| **Unidades** | **Descripción** | **Precio (€)** |
| 1 | Precio 1: Decisión del alcance | 500,00 € |
| 1 | Precio 2: Elección de tecnologías | 500,00 € |
| 1 | Precio 3: Diseño de la arquitectura | 500,00 € |
| 1 | Precio 4: Desarrollo del programa | 6.000,00 € |
| 1 | Precio 5: Desarrollo de la interfaz a la BD | 1.560,00 € |
| 1 | Precio 6: Creación y configuración del entorno | 360,00 € |
| 1 | Precio 7: Integración de las partes | 840,00 € |
| 1 | Precio 8: Testing y corrección de errores I | 3.000,00 € |
| 1 | Precio 9: Obtención de programas | 500,00 € |
| 1 | Precio 10: Ejecución del programa | 500,00 € |
| 1 | Precio 11: Testing y corrección de errores II | 2.000,00 € |
| 1 | Precio 12: Análisis de dataset | 1.500,00 € |
| 1 | Precio 13: Testing y corrección de errores III | 1.200,00 € |
| 1 | Precio 14: Documentación del proyecto | 3.000,00 € |
|  | **Precio total** | 21.960,00 € |
|  | Costes indirectos (10%) | 2.196,00 € |
|  | **Presupuesto total (directos + indirectos)** | **24.156,00 €** |

Tabla 266: Conceptos de los costes indirectos

|  |  |  |
| --- | --- | --- |
| **Conceptos** | | |
| Electricidad | | |
| Internet | | |
| Comunicaciones | | |
| Dietas | | |
| Transporte | | |
| Limpieza y mantenimiento | | |
|  |  |  |

# Referencias

|  |  |
| --- | --- |
| [1] | Python 3.12.3, «The Python Standard Library, ast - Abstract Syntax Trees,» [En línea]. Available: https://docs.python.org/3/library/ast.html. [Último acceso: Junio 2024]. |
| [2] | E. Gamma, R. Helm, R. Johnson, and J. Vlissides, «Design Patterns: Abstraction and Reuse of Object-Oriented Design,» de *European Conf. on Object-Oriented Programming*, 1993. |
| [3] | Van Rossum, G. and Drake Jr, F.L., «Python tutorial,» de *Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands*, 1995. |
| [4] | Alfred, V. Aho, S. Lam Monica, and D. Ullman Jeffrey, Compilers Principles, Techniques and Tools, Pearson Education, 2007. |
| [5] | Joglekar, M., Garcia-Molina, H. and Parameswaran, A, «Interactive data exploration with smart drill-down,» *IEEE Transactions on Knowledge and Data Engineering,* vol. 31, nº 1, pp. 46-60, 2017. |
| [6] | Adams, J., Hayunga, D., Mansi, S., Reeb, D. and Verardi, V., «Identifying and treating outliers in finance,» *Financial Management,* vol. 48, nº 2, pp. 345-384, 2019. |
| [7] | Tukey, J.W., Exploratory data analysis. Vol. 2., Reading, MA: Addison-wesley, 1977. |
| [8] | Hubert, M. and Vandervieren, E., «An adjusted boxplot for skewed distributions,» *Computational statistics & data analysis,* vol. 52, nº 12, pp. 5186-5201, 2008. |
| [9] | Liu, F.T., Ting, K.M. and Zhou, Z.H., «Isolation forest,» *2008 eighth ieee international conference on data mining,* pp. 413-422, 2008. |

# Anexos

## Diseño de la Base de Datos



Figura 3: Diagrama entidad relación con las tablas de la base de datos

## Dominio de las características

### StatementCategory

Return, Delete, AssignmentStmt, TypeAlias, AugmentedAssignment, AnnotatedAssignment, For, AsyncFor, If, While, With, AsyncWith, Match, Raise, Try, Assert, Global, NonLocal, Pass, Break, Continue, ExceptHandler, Import, ImportFrom.

### StatementRole

Module, IfBody, IfElseBody, FunctionDefBody, AsyncFunctionDefBody, MethodDefBody, AsyncMethodDefBody, ClassDefBody, ForBody, ForElseBody, AsyncForBody, AsyncForElseBody, WithBody, WhileBody, WhileElseBody, ExceptBody, AsyncWithBody, TryBody, TryElseBody, TryFinallyBody, TryHandler, TryHandlerStar, CaseBody.

### ExpressionCategory

Logical, AssignmentExp, Arithmetic, Pow, Shift, BWLogical, MatMult, UnaryArithmetic, UnaryNot, UnaryBWNot, Lambda, Ternary, SetLiteral, ListLiteral, TupleLiteral, DictionaryLiteral, ListComprehension, SetComprehension, DictComprehension, GeneratorComprehension, Await, Yield, YieldFrom, Compare, Call, FString, FormattedValue, IntLiteral, FloatLiteral, ComplexLiteral, NoneLiteral, BoolLiteral, StringLiteral, EllipsisLiteral, Dot, Variable, Slice, Indexing, Star, NoneType.

### ExpressionRole

Module, FuncDecorator, FuncBody, ReturnType, ClassBase, ClassDecorator, MethodBody, ClassBody, Return, Delete, AssignLHS, AssignRHS, TypeAliasLHS, TypeAliasRHS, AugmentedAssignmentLHS, AugmentedAssignmentRHS, VarDefVarName, VarDefType, VarDefInitValue, ForElement, ForEnumerable, ForBody, ForElseBody, AsyncForElement, AsyncForEnumerable, AsyncForBody, AsyncForElseBody, WhileCondition, WhileBody, WhileElseBody, IfCondition, IfBody, IfElseBody, WithElement, WithAs, WithBody, AsyncWithElement, AsyncWithAs, AsyncWithBody, MatchCondition, CaseCondition, CaseGuard, CaseBody, Raise, RaiseFrom, TryBody, ExceptType, ExceptBody, TryElse, FinallyBody, AssertCondition, AssertMessage, Logical, AssignExpLHS, AssignExpRHS, Arithmetic, Pow, Shift, BWLogical, MatMult, LambdaBody, TernaryCondition, TernaryIfBody, TernaryElseBody, SetLiteral, ListLiteral, TupleLiteral, DictionaryLiteralKey, DictionaryLiteralValue, ComprehensionElement, ComprehensionTarget, ComprehensionIter, ComprehensionIf, Await, Yield, YieldFrom, Relational, Is, In, CallFuncName, CallArg, FString, Dot, Slice, Indexing, Star, TypeAnnotation, DefaultParamValue, TypeVar, FormattedFormat, AugmentedAssigmentLHS, Compare, FormattedValue, AugmentedAssigmentRHS, TryElseBody, ComprenhensionElement.

## Resultados Detección de Anomalías

En la Sección **¡Error! No se encuentra el origen de la referencia.** describimos las instancias catalogadas como anómalas más relevantes para nuestro estudio. A continuación, enumeramos, por conjunto de datos estudiado, los valores de cada una de las características que hacen que sean detectados como anómalos. En cada caso, se muestran los valores teniendo en cuenta todos los datos, solo con nivel de experiencia *Beginner* y solo con nivel de experiencia *Expert*.

### Programas

* Contiene subdirectorios con código (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
* Contiene paquetes (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
* Contiene código en el directorio raíz (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es true. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
* Número de módulos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 143,7. \*\*\*\*\* 3294 \*\*\*\*\*
  + *Beginner*: Se detecta como anómalo cuando es superior a 148,39.
  + *Expert*: Se detecta como anómalo cuando es superior a 1764,39.
* Número de subdirectorios con código (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0. \*\*\*\*\*\* 238 \*\*\*\*\*\*
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 145.
* Número de paquetes (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0. \*\*\*\*\*\*\*\* 1297 \*\*\*\*\*\*\*\*
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 172,5.
* Media de definiciones por módulo (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 8,57.
  + *Beginner*: Se detecta como anómalo cuando es superior a 8.
  + *Expert*: Se detecta como anómalo cuando es inferior a 1,41.
* Proporción de definiciones de clases (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,82.
* Proporción de definiciones de funciones (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: Se detecta como anómalo cuando es inferior a 0,43.
* Proporción de definiciones de enumeraciones (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0**.**
  + *Expert*: Se detecta como anómalo cuando es superior a 0,017.

### Modulos

* Contiene comentario de módulo (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene punto de entrada (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Número de clases (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 76.
  + *Beginner*: Se detecta como anómalo cuando es superior a 72,7.
  + *Expert*: Se detecta como anómalo cuando es superior a 73,08.
* Número de funciones (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 185.
  + *Beginner*: Se detecta como anómalo cuando es superior a 185,2.
  + *Expert*: Se detecta como anómalo cuando es superior a 143.
* Media de sentencias en el cuerpo de las funciones (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 17,6.
  + *Beginner*: Se detecta como anómalo cuando es superior a 19,4.
  + *Expert*: Se detecta como anómalo cuando es superior a 41,9.
* Media de sentencias en el cuerpo de los métodos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 37,3.
  + *Beginner*: Se detecta como anómalo cuando es superior a 5,4.
  + *Expert*: Se detecta como anómalo cuando es superior a 59.
* Convención de nombrado (Nominal): En todos los casos esta variable toma 6 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,033%.
  + Todos: El valor predominante que toma esta variable es el de SnakeCase. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante que toma esta variable es el de Lower. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante que toma esta variable es el de SnakeCase. No se detectaron valores anómalos para esta característica.
* Proporción de sentencias globales (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,83.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,625.
* Proporción de expresiones globales (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,5.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0,8.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,36.
* Proporción de definiciones de clases (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Proporción de definiciones de funciones (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Proporción de definiciones de enumeraciones (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de anotaciones de tipos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,96.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.

### Imports

* Número de imports (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 123,8.
  + *Beginner*: Se detecta como anómalo cuando es superior a 20.
  + *Expert*: Se detecta como anómalo cuando es superior a 30.
* Media de módulos importados (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 4.
  + *Beginner*: Se detecta como anómalo cuando es superior a 4.
  + *Expert*: Se detecta como anómalo cuando es superior a 4.
* Media de módulos importados con un From (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 6,28.
  + *Beginner*: Se detecta como anómalo cuando es superior a 4,57.
  + *Expert*: Se detecta como anómalo cuando es superior a 19,7.
* Media de módulos importados con un As (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de módulos importados con un Import simple (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Proporción de módulos importados con un Import From (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Proporción de imports locales (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: No se detectaron valores anómalos para esta característica.

### Definiciones de clases

* Si es una definición de enumerado (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene anotación de tipo genérica (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene comentario de clase (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene una meta clase (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Número de caracteres del nombre (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 45.
  + *Beginner*: Se detecta como anómalo cuando es superior a 29.
  + *Expert*: Se detecta como anómalo cuando es superior a 49.
* Número de decoradores (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Número de métodos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 17.
  + *Beginner*: Se detecta como anómalo cuando es superior a 21.
  + *Expert*: Se detecta como anómalo cuando es superior a 17.
* Número de clases base (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 1.
  + *Beginner*: Se detecta como anómalo cuando es superior a 1.
  + *Expert*: Se detecta como anómalo cuando es superior a 4.
* Media de sentencias en el cuerpo de los métodos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 13.
  + *Beginner*: Se detecta como anómalo cuando es superior a 10,7.
  + *Expert*: Se detecta como anómalo cuando es superior a 14.
* Número de sentencias en el cuerpo de la clase (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 22.
  + *Beginner*: Se detecta como anómalo cuando es superior a 26.
  + *Expert*: Se detecta como anómalo cuando es superior a 18.
* Número de keywords (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Altura (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 1.
  + *Beginner*: Se detecta como anómalo cuando es superior a 1.
  + *Expert*: Se detecta como anómalo cuando es superior a 1.
* Proporción de asignaciones en el cuerpo de la clase (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,72.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0,57.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,625.
* Proporción de expresiones en el cuerpo de la clase (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,625.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0,83.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,66.
* Proporción de anotaciones de tipo (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,8.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,83.
* Proporción de métodos privados (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de métodos mágicos (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Proporción de métodos asíncronos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de métodos de clase (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de métodos estáticos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de métodos abstractos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de métodos de propiedad (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Convención de nombrado (Nominal):
  + Todos: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de CamelUp. No se detectaron valores anómalos para esta característica.
  + *Beginner*: En este caso la variable toma 5 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,04%. El valor predominante que toma esta variable es el de CamelUp. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores CamelLow y Discard.
  + *Expert*: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de CamelUp. No se detectaron valores anómalos para esta característica.

### Definiciones de funciones

* Si es una función privada (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es una función mágica (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. El valor true es anómalo.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es una función asíncrona (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene anotación de tipo de retorno (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene comentario de función (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Número de caracteres del nombre (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 50.
  + *Beginner*: Se detecta como anómalo cuando es superior a 41.
  + *Expert*: Se detecta como anómalo cuando es superior a 62.
* Número de sentencias en el cuerpo de la función (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 18.
  + *Beginner*: Se detecta como anómalo cuando es superior a 14.
  + *Expert*: Se detecta como anómalo cuando es superior a 18. \*\*\*\* > 100 \*\*\*\*
* Número de decoradores (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 4.
* Altura (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 1.
  + *Beginner*: Se detecta como anómalo cuando es superior a 1.
  + *Expert*: Se detecta como anómalo cuando es superior a 1.
* Proporción de sentencias en el cuerpo de la función que son expresiones (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,83.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0,89.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,83.
* Proporción de anotaciones de tipos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,83.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Convención de nombrado (Nominal): En todos los casos esta variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%.
  + Todos: El valor predominante que toma esta variable es el de SnakeCase. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante que toma esta variable es el de SnakeCase. Los valores Discard y Upper son anómalos.
  + *Expert*: El valor predominante que toma esta variable es el de SnakeCase. No se detectaron valores anómalos para esta característica.

### Definiciones de métodos

* Si es un método de clase (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método estático (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método constructor (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método abstracto (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método de propiedad (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método wrapper (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método cacheado (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método privado (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método mágico (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es un método asíncrono (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. El valor true es anómalo.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene anotación de tipo de retorno (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene un comentario de función (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Número de caracteres del nombre (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 40.
  + *Beginner*: Se detecta como anómalo cuando es superior a 32.
  + *Expert*: Se detecta como anómalo cuando es superior a 44.
* Número de sentencias en el cuerpo (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 76.
  + *Beginner*: Se detecta como anómalo cuando es superior a 35,1.
  + *Expert*: Se detecta como anómalo cuando es superior a 51,7.
* Número de decoradores (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Altura (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 2.
  + *Beginner*: Se detecta como anómalo cuando es superior a 2.
  + *Expert*: Se detecta como anómalo cuando es superior a 2.
* Proporción de sentencias en el cuerpo que son expresiones (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0,83.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Proporción de anotaciones de tipos (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: No se detectaron valores anómalos para esta característica.
* Convención de nombrado (Nominal):
  + Todos: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de SnakeCase. El valor Discard es anómalo.
  + *Beginner*: En este caso la variable toma 6 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,033%. El valor predominante que toma esta variable es el de SnakeCase. No se detectaron valores anómalos para esta característica. Esta variable nunca toma el posible valor Discard.
  + *Expert*: En este caso la variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%. El valor predominante que toma esta variable es el de SnakeCase. El valor Discard es anómalo.

### Sentencias

* Siendo una sentencia de la categoría correspondiente, si tiene una cláusula else (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Altura (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 10.
  + *Beginner*: Se detecta como anómalo cuando es superior a 10.
  + *Expert*: Se detecta como anómalo cuando es superior a 10.
* Profundidad (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 13.
  + *Beginner*: Se detecta como anómalo cuando es superior a 13.
  + *Expert*: Se detecta como anómalo cuando es superior a 13.
* Número de sentencias en el cuerpo de la sentencia, de haberlo. (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 49.
  + *Beginner*: Se detecta como anómalo cuando es superior a 26,5.
  + *Expert*: Se detecta como anómalo cuando es superior a 72,4.
* Categoría sintáctica (Nominal):
  + Todos: En este caso la variable toma 22 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0090%. El valor predominante que toma esta variable es el de AssingmentStmt. Los valores Match y Nonlocal son anómalos. Esta variable nunca toma los posibles valores ExceptHandler y TypeAlias.
  + *Beginner*: En este caso la variable toma 18 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,01%. El valor predominante que toma esta variable es el de AssingmentStmt. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores AsyncWith, AsyncFor, Match, TypeAlias, ExceptHandler y Nonlocal.
  + *Expert*: En este caso la variable toma 22 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0090%. El valor predominante que toma esta variable es el de AssingmentStmt. El valor Match es anómalo. Esta variable nunca toma los posibles valores ExceptHandler y TypeAlias.
* Categoría sintáctica del padre (Nominal):
  + Todos: En este caso la variable toma 12 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,016%. El valor predominante que toma esta variable es el de FunctionDef. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores Raise, Return, Import, ImportFrom, Global, Nonlocal, AnnotatedAssignment, Pass, AssignmentStmt, Break, Delete, Continue, Assert, AugmentedAssignment, AsyncFor y TypeAlias.
  + *Beginner*: En este caso la variable toma 10 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,02%. El valor predominante que toma esta variable es el de FunctionDef. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores Raise, Return, Import, ImportFrom, Global, Nonlocal, AnnotatedAssignment, Pass, AssignmentStmt, Break, Delete, Continue, Assert, AugmentedAssignment, TypeAlias, AsyncWith, AsyncFor y Match.
  + *Expert*: En este caso la variable toma 12 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,016%. El valor predominante que toma esta variable es el de FunctionDef. No se detectaron valores anómalos para esta característica. Esta variable nunca toma los posibles valores Raise, Return, Import, ImportFrom, Global, Nonlocal, AnnotatedAssignment, Pass, AssignmentStmt, Break, Delete, Continue, Assert, AugmentedAssignment, AsyncFor y TypeAlias.
* Rol de la sentencia (Nominal):
  + Todos: En este caso la variable toma 20 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,01%. El valor predominante que toma esta variable es el de MethodDefBody. El valor WhileElseBody es anómalo. Esta variable nunca toma los posibles valores TryHandler, AsyncForElseBody y TryHandlerStar.
  + *Beginner*: En este caso la variable toma 17 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,011%. El valor predominante que toma esta variable es el de IfBody. El valor AsyncMethodDefBody es anómalo. Esta variable nunca toma los posibles valores TryHandler, AsyncForElseBody, TryHandlerStar, AsyncForBody, CaseBody y AsyncWithBody.
  + *Expert*: En este caso la variable toma 20 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,01%. El valor predominante que toma esta variable es el de MethodDefBody. Los valores WhileElseBody y ForElseBody es anómalo. Esta variable nunca toma los posibles valores TryHandler, AsyncForElseBody y TryHandlerStar.
* Categoría sintáctica del primer hijo (Nominal):
  + Todos: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0058%. El valor predominante que toma esta variable es el de Variable. Los valores Shift, SetLiteral, SetComprehension, Pow, UnaryBWNot, MatMult y AssignmentExp son anómalos. Esta variable nunca toma los posibles valores FormattedValue, Star, YieldFrom, EllipsisLiteral, Yield, Parameter y Slice.
  + *Beginner*: En este caso la variable toma 30 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0066%. El valor predominante que toma esta variable es el de Variable. Los valores ComplexLiteral, Shift, Pow, Await y SetComprehension son anómalos.
  + *Expert*: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0058%. El valor predominante que toma esta variable es el de Variable. Los valores MatMult, UnaryBWNot, Shift, Pow, GeneratorComprehension, AssignmentExp, SetLiteral y SetComprehnsion son anómalos. Esta variable nunca toma los valores FormattedValue, Star, YieldFrom, EllipsisLiteral, Yield, Parameter y Slice.
* Categoría sintáctica del segundo hijo (Nominal):
  + Todos: En este caso la variable toma 36 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0055%. El valor predominante que toma esta variable es el de None. Los valores EllipsisLiteral, YieldFrom, Yield y UnaryBWNot son anómalos. Esta variable nunca toma los posibles valores AssignmentExp, FormattedValue, Star, Parameter y Slice.
  + *Beginner*: En este caso la variable toma 33 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,006%. El valor predominante que toma esta variable es el de None. Los valores Yield, Await y UnaryBWNot son anómalos. Esta variable nunca toma los posibles valores YieldFrom, AssignmentExp, MatMult, Star, Parameter, EllipsisLiteral, FormattedValue y Slice.
  + *Expert*: En este caso la variable toma 36 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0055%. El valor predominante que toma esta variable es el de None. Los valors EllipsisLiteral, Yield, YieldFrom y UnaryBWNot son anómalos. Esta variable nunca toma los posibles valores AssignmentExp, FormattedValue, Star, Parameter y Slice.
* Categoría sintáctica del tercer hijo (Nominal):
  + Todos: En este caso la variable toma 30 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0066%. El valor predominante que toma esta variable es el de None. Los valores Pow, ComplexLiteral, UnaryNot, SetComprehension, BWLogical, FString, GeneratorComprehension, Compare, SetLiteral, UnaryArithmentic, Logical, Lambda, DictComprehension, Await, ListComprehension, Arithmetic y Ternary son anómalos. Esta variable nunca toma los posibles valores AssignmentExp, Shift, FormattedValue, Star, MatMult, YieldFrom, Yield, EllipsisLiteral, UnaryBWNot, Parameter y Slice.
  + *Beginner*: En este caso la variable toma 23 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0087%. El valor predominante que toma esta variable es el de None. Los valores SetLiteral, FloatLiteral, UnaryArithmetic, Logical, BWLogical, BoolLiteral, ListComprehension, DictComprehension, GeneratorComprehension, Ternary, Lambda y Arithmetic son anómalos. Esta variable nunca toma los valores UnaryBWNot, YieldFrom, Shift, AssignmentExp, MatMult, Star, Parameter, Pow, Yield, SetComprehension, ComplexLiteral, FString, UnaryNot, EllipsisLiteral, Compare, FormatterValue, Await y Slice.
  + *Expert*: En este caso la variable toma 28 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0071%. El valor predominante que toma esta variable es el de None. Los valores ComplexLiteral, Pow, UnaryNot, Lambda, SetComprehension, FString, SetLiteral, UnaryArithmetic, Compare, Logical, DictComprehension, Arithmetic, TupleLiteral, ListComprehension y Ternary son anómalos. Esta variable nunca toma los posibles valores YieldFrom, EllipsisLiteral, Star, Yield, FormattedValue, UnaryBWNot, Slice, AssignmentExp, Parameter, MatMult, GeneratorComprehension, BWLogical y Shift.

### Cases

* Número de cláusulas case en el Match. (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 10.
* Número guards. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número medio de sentencias en el cuerpo de los case. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número de cases del tipo MatchValue. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número de cases del tipo MatchSingleton. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número de cases del tipo MatchSequence. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número de cases del tipo MatchMapping. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número de cases del tipo MatchStar. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número de cases del tipo MatchOr. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.
* Número de cases del tipo MatchClass. (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
* Número de cases del tipo MatchAs. (Numérica):
  + Todos: No se detectaron valores anómalos para esta característica.

### Handlers

* Contiene un handler star, es decir, es un TryStar (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene un handler que capture todas las excepciones (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene una cláusula finally (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Número de handlers. (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
  + *Beginner*: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
  + *Expert*: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
* Número medio de sentencias en el cuerpo de los handlers. (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
  + *Beginner*: Se detecta como anómalo cuando es superior a 1 o es inferior a 1.
  + *Expert*: Se detecta como anómalo cuando es superior a 5.

### Expresiones

* Altura (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 12.
  + *Beginner*: Se detecta como anómalo cuando es superior a 12.
  + *Expert*: Se detecta como anómalo cuando es superior a 12.
* Profundidad (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 4.
  + *Beginner*: Se detecta como anómalo cuando es superior a 4.
  + *Expert*: Se detecta como anómalo cuando es superior a 4.
* Altura (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 12.
  + *Beginner*: Se detecta como anómalo cuando es superior a 12.
  + *Expert*: Se detecta como anómalo cuando es superior a 12.
* Profundidad (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 4.
  + *Beginner*: Se detecta como anómalo cuando es superior a 4.
  + *Expert*: Se detecta como anómalo cuando es superior a 4.
* Categoría sintáctica (Nominal):
  + Todos: En este caso la variable toma 39 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0051%. El valor predominante que toma esta variable es el de Variable. Los valores AssignmentExp, SetComprehension, MatMult, YieldFrom y UnaryBWNot son anómalos. Esta variable nunca toma el posible valor NoneType.
  + *Beginner*: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de Variable. Los valores AssignmentExp, SetComprehension y Await son anómalos. Esta variable nunca toma los posibles valores MatMult y NoneType.
  + *Expert*: En este caso la variable toma 39 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0051%. El valor predominante que toma esta variable es el de Variable. Los valores AssingmentExp, UnaryBWNot, MatMult y YieldFrom son anómalos. Esta variable nunca toma el posible valor NoneType.
* Categoría sintáctica del padre (Nominal):
  + Todos: En este caso la variable toma 51 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0039%. El valor predominante que toma esta variable es el de Call. Los valores AsyncWith, AssignmentExp, Match, UnaryBWNot y MatMult son anómalos. Esta variable nunca toma los posibles valores Import, Break, Nonlocal, StringLiteral, IntLiteral, FloatLiteral, Global, Variable, NoneLiteral, EllipsisLiteral, Continue, NoneType, TypeAlias, ComplexLiteral, BoolLiteral, ImportFrom y Pass.
  + *Beginner*: En este caso la variable toma 47 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0042%. El valor predominante que toma esta variable es el de Call. Los valores AssignmentExp y Await son anómalos. Este valor nunca toma los posibles valores NoneLiteral, Import, Global, FloatLiteral, NoneType, EllipsisLiteral, TypeAlias, Match, Pass, Continue, Nonlocal, ComplexLiteral, ImportFrom, MatMult, Variable, Break, StringLiteral, BoolLiteral, IntLiteral, AsyncFor y AsyncWith.
  + *Expert*: En este caso la variable toma 51 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0039%. El valor predominante que toma esta variable es el de Call. Los valores AssignmentExp, UnaryBWNot, Match y YieldFrom son anómalos. Esta variable nunca toma los posibles valores ComplexLiteral, Global, ImportFrom, Continue, StringLiteral, NoneLiteral, Nonlocal, FloatLiteral, EllipsisLiteral, Import, NoneType, Break, IntLiteral, Pass, TypeAlias, BoolLiteral y Variable.
* Categoría sintáctica del primer hijo (Nominal):
  + Todos: En este caso la variable toma 39 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0051%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, SetComprehension, DictComprehension, NoneType, Shift, Lambda, GeneratorComprehension, MatMult, Await, UnaryBWNot, Star y EllipsisLiteral son anómalos. Esta variable nunca toma los posibles valores YieldFrom, Yield y Parameter.
  + *Beginner*: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0059%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, SetComprehension, NoneType, Star, GeneratorComprehension, ListComprehension, UnaryBWNot, Shift, FString y DictionaryLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, DictComprehension, Await, Yield, YieldFrom, Slice, Lambda y MatMult.
  + *Expert*: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de None. Los valores UnaryBWNot, SetComprehension, DictComprehension, GeneratorComprehension, Lambda, Await, Star, MatMult, EllipsisLiteral y Shift son anómalos. Esta variable nunca toma los posibles valores YieldFrom, Parameter, YIeld y AssignmentExp.
* Categoría sintáctica del segundo hijo (Nominal):
  + Todos: En este caso la variable toma 37 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0054%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, Await, SetComprehension, Shift, DictComprehension, UnaryBWNot y SetLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, YieldFrom, MatMult, NoneType y Yield.
  + *Beginner*: En este caso la variable toma 36 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0055%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, Shift, DictComprehension, SetLiteral, SetComprehension, UnaryBWNot y EllipsisLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, Await, Yield, YieldFrom, NoneType y MatMult.
  + *Expert*: En este caso la variable toma 37 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0054%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, SetComprehension, UnaryBWNot, Await, Shift y DictComprehension son anómalos. Esta variable nunca toma los posibles valores MatMult, YieldFrom, NoneTyp, Parameter y Yield.
* Categoría sintáctica del tercer hijo (Nominal):
  + Todos: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de None. Los valores SetComprehension, UnaryBWNot, AssignmentExp, Shift, EllipsisLiteral, Await, DictComprehension, GeneratorComprehension, Pow, SetLiteral, ListComprehension, BWLogical y NoneType son anómalos. Esta variable nunca toma los posibles valores YieldFrom, Yield, Parameter y MatMult.
  + *Beginner*: En este caso la variable toma 32 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,00625%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, DictComprehension, Pow, NoneType, Setliteral, ListComprehension, GeneratorComprehension, ComplexLiteral, FString, Ternary, Lambda y FloatLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, Await, Shift, Yield, UnaryBWNot, YieldFrom, EllipsisLiteral, Slice, SetComprehension y MatMult.
  + *Expert*: En este caso la variable toma 38 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0052%. El valor predominante que toma esta variable es el de None. Los valores AssignmentExp, SetComprehension, UnaryBWNot, Shift, GeneratorComprehension, EllipsisLiteral, Await, DictComprehension, BWLogical, Pow, SetLiteral y ListComprehension son anómalos. Esta variable nunca toma los posibles valores YieldFrom, Parameter, Yield y MatMult.
* Categoría sintáctica del cuarto hijo (Nominal):
  + Todos: En este caso la variable toma 35 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0057%. El valor predominante que toma esta variable es el de None. Los valores Await, SetComprehension, EllipsisLiteral, Shift, GeneratorComprehension, DictComprehension, BWLogical, Pow, SetLiteral, ListComprehension, ComplexLiteral, Slice, Star, Ternary, Lambda, FString y UnaryNot son anómalos. Esta variable nunca toma los posibles valores Parameter, YieldFrom, MatMult, UnaryBWNot, NoneType, AssignmentExp y Yield.
  + *Beginner*: En este caso la variable toma 29 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0069%. El valor predominante que toma esta variable es el de None. Los valores DictComprehension, ListComprehension, SetLiteral, GeneratorComprehension, ComplexLiteral, BWLogical, FString, Ternary, FloatLiteral, Lambda, Star, UnaryNot y DictionaryLiteral son anómalos. Esta variable nunca toma los posibles valores Parameter, Await, Shift, Yield, UnaryBWNot, YieldFrom, EllipsisLiteral, Slice, Pow, AssignmentExp, NoneType, SetComprehension y MatMult.
  + *Expert*: En este caso la variable toma 34 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0059%. El valor predominante que toma esta variable es el de None. Los valores SetComprehension, Await, EllipsisLiteral, Shift, DictComprehension, BWLogical, Star, SetLiteral, Pow, ComplexLiteral, ListComprehension, Slice, UnaryNot, Ternary, Lambda, FString y Logical son anómalos. Esta variable nunca toma los posibles valores MatMult, GeneratorComprehension, YieldFrom, NoneType, Parameter, UnaryBWNot, Yield y AssignmentExp.
* Rol de la expresión en su padre (Nominal):
  + Todos: En este caso la variable toma 79 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0025%. El valor predominante que toma esta variable es el de CallArg. Los valores CaseBody, WhileElseBody, MatchCondition, AsyncWithBody, AssignExpLHS, AssignExpRHS, ForElseBody, AsyncWithAs, CaseCondition, AsyncForBody, AsyncWithElement, AsyncForEnumerable y AsyncForElement son anómalos. Esta variable nunca toma los posibles valores TryElse, ComprehensionElement, AsyncForElseBody, MatMult, DefaultParamValue, TypeVar, In, Shift, AugmentedAssignmentLHS, TypeAliasLHS, CaseGuard, Pow, Is, AugmentedAssignmentRHS, TypeAliasRHS y BWLogical.
  + *Beginner*: En este caso la variable toma 70 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0028%. El valor predominante que toma esta variable es el de TupleLiteral. Los valores WhileElseBody, ClassDecorator, AssignExpRHS, AssignExpLHS, ForElseBody y FormattedFormat son anómalos. Esta variable nunca toma los posibles valores AsyncWithElement, AugmentedAssignmentLHS, Shift, DefaultParamValue, MatchCondition, AsyncForBody, BWLogical, TryElse, In, Pow, TypeVar, AsyncWithAs, CaseBody, MatMult, CaseCondition, ComprehensionElement, AugmentedAssignmentRHS, TypeAnnotation, AsyncForEnumerable, CaseGuard, AsyncForElseBody, TypeAliasRHS, TypeAliasLHS, AyncWithBody, Is y AsyncForElement.
  + Expert: En este caso la variable toma 79 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,0025%. El valor predominante que toma esta variable es el de CallArg. Los valores WhileElseBody, CaseBody, ForElseBody, MatchCondition, AsyncWithBody, AssignExpRHS, AssignExpLHS, TryElseBody y AsyncWithAs son anómalos. Esta variable nunca toma los posibles valores TypeVar, MatMult, BWLogical, CaseGuard, AugmentedAssignmentLHS, TryElse, TypeAnnotation, TypeAliasRHS, AsyncForElseBody, Is, TypeAliasLHS, Pow, DefaultParamValue, AugmentedAssignmentRHS, ComprehensionElement, In y Shift.

### Comprehensions

* Si es asíncrono (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. El valor true es anómalo.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. El valor true es anómalo.
* Número de sentencias condicionales (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: No se detectaron valores anómalos para esta característica.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Número de generadores (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 1.
  + *Beginner*: Se detecta como anómalo cuando es superior a 1.
  + *Expert*: Se detecta como anómalo cuando es superior a 1.
* Categoría sintáctica (Nominal): En todos los casos esta variable toma 4 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,05%.
  + Todos: El valor predominante que toma esta variable es el de ListComprehension. No se han detectado valores anómalos para esta característica.
  + *Beginner*: El valor predominante que toma esta variable es el de GeneratorComprehension. No se han detectado valores anómalos para esta característica.
  + *Expert*: El valor predominante que toma esta variable es el de ListComprehension. No se han detectado valores anómalos para esta característica.

### Invocaciones a funciones

* Número de argumentos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 5.
  + *Beginner*: Se detecta como anómalo cuando es superior a 5.
  + *Expert*: Se detecta como anómalo cuando es superior a 5.
* Proporción de argumentos pasados por nombre (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de argumentos double star (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.

### Cadenas formateadas

* Número de elementos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 10.
  + *Beginner*: Se detecta como anómalo cuando es superior a 10.
  + *Expert*: Se detecta como anómalo cuando es superior a 10.
* Proporción de constantes (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,91 o cuando es inferior a 0,25.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0,91 o cuando es inferior a 0,25.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,91 o cuando es inferior a 0,25.
* Proporción de expresiones (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0,75 o cuando es inferior a 0,08.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0,75 o cuando es inferior a 0,08.
  + *Expert*: Se detecta como anómalo cuando es superior a 0,75 o cuando es inferior a 0,08.

### Variables

* Si es privada (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Si es mágica (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Número de caracteres (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 28.
  + *Beginner*: Se detecta como anómalo cuando es superior a 20.
  + *Expert*: Se detecta como anómalo cuando es superior a 28.
* Convención de nombrado (Nominal): En todos los casos esta variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%.
  + Todos: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.

### Vectores

* Si es homogénea (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es true. No se detectaron valores anómalos para esta característica.
* Número de elementos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 81 o cuando es inferior a 1,9.
  + *Beginner*: Se detecta como anómalo cuando es superior a 77 o cuando es inferior a 1,9.
  + *Expert*: Se detecta como anómalo cuando es superior a 2 o cuando es inferior a 2. \*\*
* Categoría sintáctica (Nominal): En todos los casos esta variable toma 4 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,05%.
  + Todos: El valor predominante que toma esta variable es el de TupleLiteral. No se han detectado valores anómalos para esta característica.
  + *Beginner*: El valor predominante que toma esta variable es el de TupleLiteral. No se han detectado valores anómalos para esta característica.
  + *Expert*: El valor predominante que toma esta variable es el de TupleLiteral. No se han detectado valores anómalos para esta característica.

### Parámetros de declaración de funciones

* Contiene al menos un var param (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Contiene al menos un keyword param (Binaria): Se considera anómalo cuando uno de los dos valores que puede tomar la variable no aparece al menos un 0,10%.
  + Todos: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es false. No se detectaron valores anómalos para esta característica.
* Número de parámetros (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 11 o inferior a 1.
  + *Beginner*: Se detecta como anómalo cuando es superior a 9 o inferior a 1.
  + *Expert*: Se detecta como anómalo cuando es inferior a 1.
* Proporción de argumentos posicionales (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de anotaciones de tipos (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de var params (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de argumentos por keyword (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Proporción de argumentos con valor por defecto (Numérica):
  + Todos: Se detecta como anómalo cuando es superior a 0.
  + *Beginner*: Se detecta como anómalo cuando es superior a 0.
  + *Expert*: Se detecta como anómalo cuando es superior a 0.
* Rol de los parámetros (Nominal): En todos los casos esta variable toma 2 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,1%.
  + Todos: El valor predominante es FunctionParams. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es FunctionParams. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es FunctionParams. No se detectaron valores anómalos para esta característica.
* Convención de nombrado más usada (Nominal): En todos los casos esta variable toma 7 valores, se considera como valor anómalo todos aquellos que tienen una frecuencia inferior a 0,028%.
  + Todos: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
  + *Beginner*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.
  + *Expert*: El valor predominante es Lower. No se detectaron valores anómalos para esta característica.

## Repositorios GitHub

En este anexo se van a listar y describir resumidamente los repositorios de *GitHub* utilizados como fuentes de código de expertos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Contribuidores | Forks | Stars | Usos |
| ray-project/ray | 974 | 5.300 | 31.607 | 15.000 |
| [certbot/certbot](https://github.com/certbot/certbot) | 478 | 3.400 | 30.992 | 1.900 |
| [run-llama/llama\_index](https://github.com/run-llama/llama_index) | 994 | 4.500 | 32.387 | 8.600 |
| [sqlmapproject/sqlmap](https://github.com/sqlmapproject/sqlmap) | 125 | 5.600 | 30.910 | X\* |
| [geekcomputers/Python](https://github.com/ComputationalReflection/PythonSourceCodeAnalysis) | 691 | 12.000 | 30.340 | X\* |
| [huggingface/pytorch-image-models](https://github.com/huggingface/pytorch-image-models) | 120 | 4.600 | 30.281 | 28.900 |
| [babysor/MockingBird](https://github.com/babysor/MockingBird) | 37 | 5.100 | 34.174 | X\* |
| [testerSunshine/12306](https://github.com/testerSunshine/12306) | 23 | 9.800 | 33.723 | X\* |
| [lm-sys/FastChat](https://github.com/lm-sys/FastChat) | 249 | 4.300 | 35.098 | 740 |
| [shadowsocks/shadowsocks](https://github.com/shadowsocks/shadowsocks) | X\* | 18.700 | 33.522 | X\* |
| [XX-net/XX-Net](https://github.com/XX-net/XX-Net) | 73 | 7.700 | 32.736 | X\* |
| [microsoft/DeepSpeed](https://github.com/microsoft/DeepSpeed) | 341 | 3.900 | 33.252 | 7.100 |
| [fxsjy/jieba](https://github.com/fxsjy/jieba) | 40 | 6.700 | 32.602 | 28.000 |
| [comfyanonymous/ComfyUI](https://github.com/comfyanonymous/ComfyUI) | 111 | 3.900 | 36.823 | X\* |
| [hankcs/HanLP](https://github.com/hankcs/HanLP) | 36 | 9.700 | 32.710 | 184 |
| [httpie/cli](https://github.com/httpie/cli) | 155 | 3.700 | 32.400 | X\* |
| [karpathy/nanoGPT](https://github.com/karpathy/nanoGPT) | 36 | 5.000 | 32.735 | X\* |
| [streamlit/streamlit](https://github.com/streamlit/streamlit) | 227 | 2.800 | 32.589 | 395.000 |

\* Esta información no consta en la descripción del repositorio a la fecha de realización del trabajo.