## Generación de tablas

Tal y como hemos mencionado, los algoritmos clásicos de minería de datos se basan en datos tabulares por lo que, para utilizar dichos algoritmos, necesitamos convertir los ASTs a tablas. Lo primero es definir los distintos tipos de nodos del AST que tengan una estructura homogénea, para poder almacenarlos en la misma tabla. Identificamos los siguientes 7 tipos de nodos: proyecto (conjunto de directorios y ficheros Python que componen un proyecto independiente), módulos (fichero Python), definiciones de clases, definición de funciones, definiciones de métodos, sentencias y expresiones y parámetros (conjunto de parámetros que se definen en una función o expresión lambda).

Para algunos de estos 7 tipos principales vamos a definir tablas auxiliares para almacenar cierta información extra. En el caso de los módulos, definiremos los imports. Para complementar la tabla de definiciones de funciones, definiremos la tabla de definiciones de métodos (funciones definidas dentro de una clase). Como tablas derivadas de sentencias vamos a definir 2, la tabla de cases (información extra relativa a las sentencias de tipo Match) y la tabla de handlers (información extra relativa a las sentencias Try y TryStar). Por último, para complementar las expresiones, vamos a definir 5 tablas: comprehensions (información relativa a generadores de listas, diccionarios, tuplas y sets), invocaciones a funciones, cadenas de texto formateadas, variables, vectores (información relativa a los literales de tipo lista, diccionario, tuplas y sets).

Además, incluiremos una última tabla que sirve para relacionar a cada elemento con su elemento padre. Esta tabla se llamará nodos y contendrá 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 los joins.

A modo de ejemplo inicial, la Tabla 8 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 los identificadores de sus tres primeros hijos.

Para realizar la traducción, tal y como se ha explicado anteriormente, hacemos uso del patrón de diseño *Visitor* [20]. Definimos un método visit para cada una de los cerca de 80 posibles tipos de nodos que presenta el árbol AST de Python. En cada uno de estos métodos visit recogemos la información relativa a cada nodo, además de propagar la inspección a sus hijos y así poder recoger información relativa al subárbol AST que tiene el nodo actual como raíz.

Dentro de cada una de las tablas que hemos comentado, exceptuando la tabla nodos, se almacena el nivel de experiencia del usuario autor de dicho nodo y el identificador de este para poder identificar los nodos escritos por un mismo autor.

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

### Proyecto

La Tabla 1 muestra la información almacenada para cada proyecto. 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 proyecto.

| **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 definiones 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) |

### Módulos

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 |
| Path | Ruta del módulo dentro del proyecto | String |
| 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) |

### 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] |
| 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) |

### Definición de clases

Respecto a la definición de clases, la Tabla 4 introduce aquellas características finales extraídas de este nodo. 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 métodos.

| **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 asincronos | [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] |
| Source code | Código fuente de la clase | String |
| 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) |

### Definiciones de funciones

Tabla 5: Características para definiciones de funciones.

| **Nombre** | **Descripción** | **Dominio** |
| --- | --- | --- |
| Name convention | La convención utilizada para el nombre de la clase. | 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] |
| Source code | Código fuente de la definición de función | String |
| 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) |

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 (el resto de los tipos nombrados en la tabla de definiciones de clases se almacenan en la tabla complementaria a esta para las definiciones de métodos), funciones privadas, funciones asíncronas y funciones mágicas. Para cada uno de estos tipos vamos a almacenar si la función es de ese tipo.

### Definiciones de métodos

Tabla 6: Características 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 class method | Si el método es un wrapper. Esto es si tiene el decorador @wraps | True or False |
| 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) |

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 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).

### Sentencias

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 definiremos un dominio complejo que se podrá consultar en el Anexo 1. Dominios complejos.

Además de esos tres campos, vamos a guardar su profundidad y su altura, su código fuente, los identificadores de sus tres primeros hijos (para poder hacer joins entre tablas) 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 and 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, AsyncWith. N/A en otro caso | True, False, N/A |
| First, second and third child’s IDs | Identificador del primer, segundo y tercer hijo de la sentencia respectivamente | Unique ID (Integer) |
| 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) |

\* Indica un dominio complejo. Este dominio se encuentra 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 |
| 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) |

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 Match (MatchValue, MatchSingleton, MatchSequence, MatchMapping, MatchClass, MatchStar, MatchAs y MatchOr), además de el número medio de sentencias en los cuerpos de las cláusulas case, el número de cases total y la proporción de guards en relación al 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 |
| 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) |

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 y, si algún handler incluye el operador estrella (si la sentencia es un TryStar, contiene un except \*\*).

### Expresiones

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 |
| Source code | Código fuente de la expresion | String |
| 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) |

\* Índica un dominio complejo. Este dominio se encuentra definido en el Anexo 9.1.

La Tabla 10 muestra la información almacenada para las expresiones. En este caso, 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, la profundidad y el código fuente.

Para las operaciones, tanto binarias como unarias, hemos dividido en varias categorías sintácticas diferentes en función del operador. Para ver más detalles al respecto de esta separación se puede consultar el Anexo 1 en el que se enumeran las diferentes categorías sintácticas de las expresiones y a que nodos de la estructura base del árbol AST de Python a la que corresponden.

En los apartados siguientes vamos a comentar las diferentes tablas auxiliares utilizadas para completar la información presente en la tabla expresiones con relación a las categorías sintácticas que necesitan dicha información extra.

### Comprehensions

Tabla11: 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 |
| 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) |

Esta es una de las tablas auxiliares a la tabla de expresiones. En este caso, contiene la información relativa a las comprehensiones. 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.

### Invocaciones a funciones

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] |
| 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) |

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

### Cadenas formateadas

La información extra 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] |
| 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) |

### Variables

En este caso, esta tabla contiene la información extra relativa a las variables. La información extra será: 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 |
| 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) |

### Vectores

Esta es la última tabla del conjunto de tablas auxiliares a expresiones. En esta recogeremos la información extra necesaria para analizar 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.

Tabla15: 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 |
| 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) |

\* Índica un dominio complejo. Este dominio se encuentra definido en el Anexo 9.1.

### Parámetros

En esta tabla, almacenaremos la información relativa a los parámetros declarados tanto en la definición de funciones como en las lambda expresiones. Para distinguir entre los parámetros que provienen de cada una de las dos 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 boleano 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 |
| 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) |

# Anexo

## Anexo 1: Dominios complejos

## Statement Category:

* Return
* AssignmentStmt
* TypeAlias
* AugmentedAssignment
* AnnotatedAssignment
* For
* If
* While
* With
* AsyncWith
* Match
* Raise
* Try
* Assert
* Global
* NonLocal
* Pass
* Break
* Continue
* ExceptHandler

Statement Role:

* Module
* FunctionDefBody
* AsyncFunctionDefBody
* MethodDefBody
* AsyncMethodDefBody
* ClassDefBody
* ForBody
* ForElseBody
* AsyncForBody
* AsyncForElseBody
* WithBody
* AsyncWithBody
* TryBody
* TryElseBody
* TryFinallyBody
* TryHandler
* TryHandlerStar
* CaseBody

ExpressionsCategory

* Logical
* AssignmentExp
* Arithmetic (BinOp with Add, Sub, Mult, Div, FloorDiv and Mod operators)
* Pow (BinOp with Pow operator)
* Shift (BinOp with LShift and RShift operators)
* BW1Logical (BinOp with BitOr, BitXor and BitAnd operators)
* MatMult (BinOp with MatMult operator)
* UnaryAritmetic (BinOp with UAdd y USub operators)
* UnaryNot (BinOp with Not operator)
* UnaryBWNot (BinOp with Invert operator)
* Lambda
* Ternary
* SetLiteral
* ListLiteral
* TupleLiteral
* DictionaryLiteral
* ListComprehension
* SetComprehension
* DictComprehension
* GeneratorComprehension
* Await
* Yield
* YieldFrom
* Compare
* Call
* FString
* IntLiteral (Constant with int value)
* FloatLiteral (Constant with float value)
* ComplexLiteral (Constant with complex value)
* NoneLiteral (Constant with None value)
* BoolLiteral (Constant with True or False value)
* StringLiteral (Constant with str value)
* EllipsisLiteral (Constant with Ellipsis value)
* Dot
* Variable
* Slice
* Indexing
* Star