# Contexto

Python es un lenguaje de programación interpretado. Esto implica que en la ejecución del código no existe una fase de generación de árboles AST previa ejecución. Para poder tener acceso a un árbol de este tipo en un lenguaje interpretado como Python vamos a utilizar la librería *ast* ([ast — Abstract Syntax Trees — Python 3.12.3 documentation](https://docs.python.org/3/library/ast.html)).

La librería *ast* se encarga de generar una estructura de árbol AST a partir de código fuente estructurada en 80 nodos diferentes. Estos 80 nodos están divididos en dos categorías principales, expresiones (26) y sentencias (28). El resto de los nodos son estructuras secundarias dependientes de alguno de los ya mencionados. A continuación, se listan todos los tipos de nodos existentes en nuestro árbol.

1. **Module**
2. sentencias
   1. **FunctionDef**
   2. **AsyncFunctionDef**
   3. **ClassDef**
   4. **Return**
   5. **Delete**
   6. **Assign**
   7. **TypeAlias**
   8. **AugAssign**
   9. **For**
   10. **AsyncFor**
   11. **While**
   12. **If**
   13. **With**
   14. **AsyncWith**
   15. **Match**
   16. **Raise**
   17. **Try**
   18. **TryStar**
   19. **Assert**
   20. **Import**
   21. **ImportFrom**
   22. **Global**
   23. **Nonlocal**
   24. **Expr**
   25. **Pass**
   26. **Break**
   27. **Continue**
3. expresiones
   1. **BollOp**
   2. **NamedExp**
   3. **BinOp**
   4. **UnaryOp**
   5. **Lambda**
   6. **IfExp**
   7. **Dict**
   8. **Set**
   9. **ListComp**
   10. **SetComp**
   11. **DictComp**
   12. **GeneratorExp**
   13. **Aeait**
   14. **Yield**
   15. **YieldFrom**
   16. **Compare**
   17. **Call**
   18. **FormattedValue**
   19. **JoinedStr**
   20. **Constant**
   21. **Attribute**
   22. **Subscript**
   23. **Starred**
   24. **Name**
   25. **List**
   26. **Tuple**
4. expresiones internas de **Subscript**
   1. **Slice**
5. contextos de expresiones
   1. **Load | Store | Del**
6. operadores booleanos
   1. **And | Or**
7. operadores binarios
   1. **Add | Sub | Mult | MatMult | Div | Mod | Pow | LShift | RShift | BitOr | BitXor | BitAnd | FloorDiv**
8. operadores unarios
   1. **Invert | Not | UAdd | USub**
9. operadores de comparación
   1. **Eq | NotEq | Lt | Lte | Gt | GtE | Is | IsNot | In | NotIn**
10. nodos derivados de los nodos **Try** y **TryStar**
    1. **ExceptHandler**
11. nodos derivados del nodo **Match**
    1. **MatchValue**
    2. **MatchSingleton**
    3. **MatchSequence**
    4. **MatchMapping**
    5. **MatchClass**
    6. **MatchStar**
    7. **MatchAs**
    8. **MatchOr­**
12. nodos secundarios
    1. **TypeIgnore**
    2. **TypeVar**
    3. **ParamSpec**
    4. **TypeVarTuple**

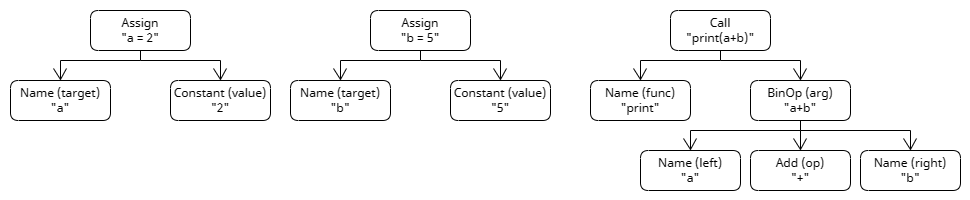
Para ejemplificar esto vamos a partir de un programa Python muy sencillo:

a = 2

b = 5

print(a+b)

El árbol AST generado para este programa de ejemplo sería el siguiente:



Podemos observar la estructura jerárquica del árbol. En cada uno de los nodos tenemos su tipo (*Assign*, *Name*, *Constant*, etc) y su rol dentro del nodo padre (*target*, *value*, *left*, etc).

El objetivo es convertir la información contenida en ese árbol en información con formato tabular. Al tener dicha información en formato tabular resultará mucho más sencillo su análisis.

# Que es un AST

Un AST o árbol de sintaxis abstracta en español es una representación de información en forma de árbol. Sirve para representar el código fuente en un lenguaje de programación, Python en este caso, de forma jerárquica estableciendo los diferentes niveles en los que se estructura dicho código.

Estos árboles no representen la totalidad de la información presente en el código fuente. Ignoran cualquier clase de símbolo o anotación no determinante en la validación y ejecución del código.

# Generación de tablas

Para el futuro análisis de la información contenida en el AST se deberá pasar esta información a un formato tabular. Lo primero será definir las tablas homogéneas en las que nos interesa almacenar la información. Para esto identificaremos las siguientes estructuras principales: proyecto (Projects), módulos (Modules), definiciones de clases (ClassDefs), definiciones de funciones (FunctionDefs), sentencias (Statements) y expresiones (Expressions). Generaremos una tabla para cada una de estas estructuras principales. Además de estas estructuras principales tendremos que almacenar la información adicional de ciertos nodos. Para los módulos almacenaremos la información relativa a los imports; para las definiciones de funciones almacenaremos las que se definan dentro de una clase (métodos) en una tabla MethodDefs y, la información relativa a los parámetros en la tabla Parameters; con respecto a las sentencias almacenaremos en una tabla Cases la información relativa a los nodos de tipo Match y, la información relativa a los nodos Try y TryStar en una tabla Handlers; por último, en lo relativo a las expresiones guardaremos por separado la información de la definición de estructuras de tipo lista (listas, diccionarios, etc) en una tabla Comprehensions, la relativa a las llamadas de funciones en una tabla CallArgs, la relativa a las cadenas formateadas en la tabla FStrings, la relativa a las variables en la tabla Variables y, lo relativo a literales de tipo lista (listas, diccionarios, etc) en la tabla Vectors.

Para la separación de la información en las 17 tablas mencionadas anteriormente utilizaremos el patrón visitor. Para esto generaremos un recorrido por cada uno de los 80 nodos, listados en el apartado [1. Contexto](#_Contexto), asignando su información a la tabla correspondiente. Más adelante se especificará la información de que nodos se guarda en cada tabla.

## Proyecto

Un proyecto abarca todos los ficheros pertenecientes a un mismo desarrollo Python. Será considerado proyecto cualquier directorio en el que haya al menos un fichero Python, independientemente de si hay más ficheros Python o subdirectorios. A continuación, se lista la información que se almacenará relativa a dichos proyectos.

## Módulos

El módulo es la estructura que representa a un fichero de código fuente Python. Cada fichero .py será almacenado enteramente en un nodo Module. A continuación, se muestra la información que se almacenará relativa a dichos modulos.

## Imports

Vamos a almacenar en esta tabla la información relativa a las sentencias Import e ImportFrom presentes en un fichero (Módulo).

## Definiciones de clases

Las definiciones de clase con únicamente los módulos ClassDef del tipo “class a():” y recogerá toda la información referente a la clase y a las sentencias que contiene.

## Definiciones de funciones

Las definiciones de funciones abarcan tanto los nodos FunctionDef, como los nodos AsyncFunctionDef. En esta tabla también se tendrán en cuenta las funciones que sean declaradas dentro de clases, aunque estas funciones tendrán una entrada especial en la tabla MethodDefs.

## Definiciones de métodos

Como ya se explicó anteriormente aquí constará la información extra necesaria con respecto a las funciones que sean definidas dentro de una clase.

## Parámetros

En esta tabla se almacenará información relativa a los parámetros de los nodos FunctionDef, AsyncFunctionDef y Lambda.

## Sentencias

En esta tabla se almacenará la información de todos los nodos listados dentro de la categoría de sentencias en el apartado [1. Contexto](#_Contexto). Algunos nodos especiales tendrán entradas en tablas auxiliares que se listarán en los apartados siguientes.

## Casos

Esta tabla registrará una entrada por cada nodo Match, almacenará la información relativa a las diferentes cláusulas “case:” del nodo.

## Manejadores

Existirá una entrada en esta tabla por cada nodo Try o TryStar. Aquí se almacenará la información de las cláusulas “except:” de estos nodos.

## Expresiones

En esta tabla se almacenará la información de todos los nodos listados dentro de la categoría de expresiones en el apartado [1. Contexto](#_Contexto). Algunos nodos especiales tendrán entradas en tablas auxiliares que se listarán en los apartados siguientes.

## Comprehensión

Las comprehensiones son los nodos ListComp, SetComp, DictComp y GeneratorExp. Se genera una entrada en esta tabla, además de en la tabla expresiones, por cada uno de estos nodos.

## Llamadas a funciones

En esta tabla se almacenarán datos extra sobre las llamadas de funciones (nodo Call), complementando la ya incluida en la tabla expresiones.

## Cadenas formateadas

La información extra de los nodos JoinedStr se almacenará en esta tabla.

## Variables

Todo lo relativo los nodos Name se almacenará en esta tabla.

## Vectores

En esta tabla se almacenará la información relativa a todos los literales de tipo lista (listas, diccionarios, etc), incluyendo los nodos List, Tuple, Dict y Set.