# 14.4 Abstract Syntax Trees in Practice

We've now reached the end of this brief chapter on abstract syntax trees, but of course the full Python programming language consists of much, much more than what we've covered! But even though a complete representation of the full Python language is far beyond the scope of this course, the work that we're doing here is *not* merely theoretical, but is actually a concrete part of the Python language itself, and tools which operate on Python programs. So we wanted to end this chapter with a brief look at some of the applications of abstract syntax trees in computer science that you'll come across in the future.

## *Python's `ast` module*

Python has a built-in module called `ast` that uses the same approach we've covered in this chapter, but cover the entire spectrum of the Python language. If you're interested in reading more about this, feel free to check out some excellent documentation at https://greentreesnakes.readthedocs.io, or an online AST exploration tool at https://python-ast-explorer.com/.

## *Code analysis*

In this chapter we focused on how an abstract syntax tree can be used to run a program, like what the Python interpreter does. However, abstract syntax trees can also be used to analyse a program's code without running it, which is known as *static program analysis*. Here are some examples:

- Check for common errors (this is what libraries like PythonTA and `pylint` do, but also software like PyCharm itself).
- Identify unused or redundant code.
- Check the types of expressions and definitions (the Python library `mypy` does this).[1]

[1] Many programming languages, but not Python, do type-checking as part of the *compilation* process, which is a step that takes your program's source code and translates it into instructions that are directly executable by your computer.

- Check for common security or efficiency problems.

## *Code manipulation and generation*

Once we can write programs that analyse code, we can write programs that modify code as well. PyCharm has a few such features built-in, that you might have already used:

- Autocompletion of variables and attributes as you type.
- “Smart” renaming of functions and variables, that automatically renames all uses of the function or variable being renamed.
- Reorganizing import statements and removing unused imports.

In Python and other programming languages, common code patterns (like “getters and setters” in Java) can be automatically generated by encoding an AST structure.

## *Transpilers*

The concept of abstract syntax trees applies to every programming language. Even though the types of allowed expressions and their exact syntax differs from language to language, abstract syntax trees provide enough structure that we can usually transform an abstract syntax tree in one language to an equivalent abstract syntax tree in another.

This allows us to develop tools to translate code from one programming language to another, or between different versions of the same programming language. As one famous example, Javacript is the common programming language for running a website’s code directly in a user’s web browser. However, different web browsers support different versions of the JavaScript language, and often lag behind the latest Javascript version that web programmers might want to use.

To get around this, tools like Babel translate between newer versions of Javascript to older versions, allowing programmers to write code using the latest Javascript features but ensuring that the code that is run on their users’ web browsers adheres to a universal standard.