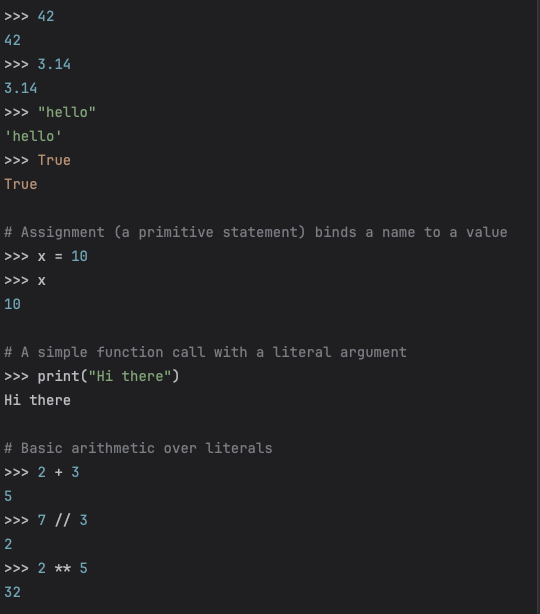
Building Programs in Python: Primitives, Combination, and Abstraction

1. Primitive expressions and statements In Python, primitives are the simplest pieces the interpreter understands directly without needing to break them down further. They include literal values (like numbers and strings), names (variables), basic operators, and single-step statements (like assignment and print).

* Primitive expressions: numbers, strings, booleans, basic arithmetic on literals.
* Primitive statements: assignments, simple function calls (e.g., print()), imports, and pass/return at their simplest use.

Interactive Python examples:

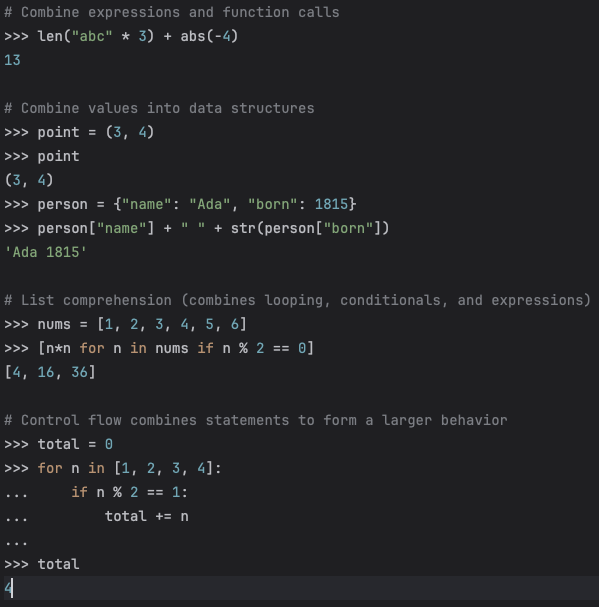


What this shows: Python can evaluate literal values directly, bind them to names with assignment, and evaluate basic operations (addition, power, integer division) without needing more structure.

1. Means of combination Combination is how we build more complex ideas by composing simpler ones. Python provides several ways to do this:

* Expression composition: combine expressions with operators and function calls.
* Data structures: group values into tuples, lists, dicts, sets.
* Control flow: combine statements with if/elif/else, while, for, and comprehensions.
* Function composition: call a function with the result of another function, chain operations.

Interactive Python examples:

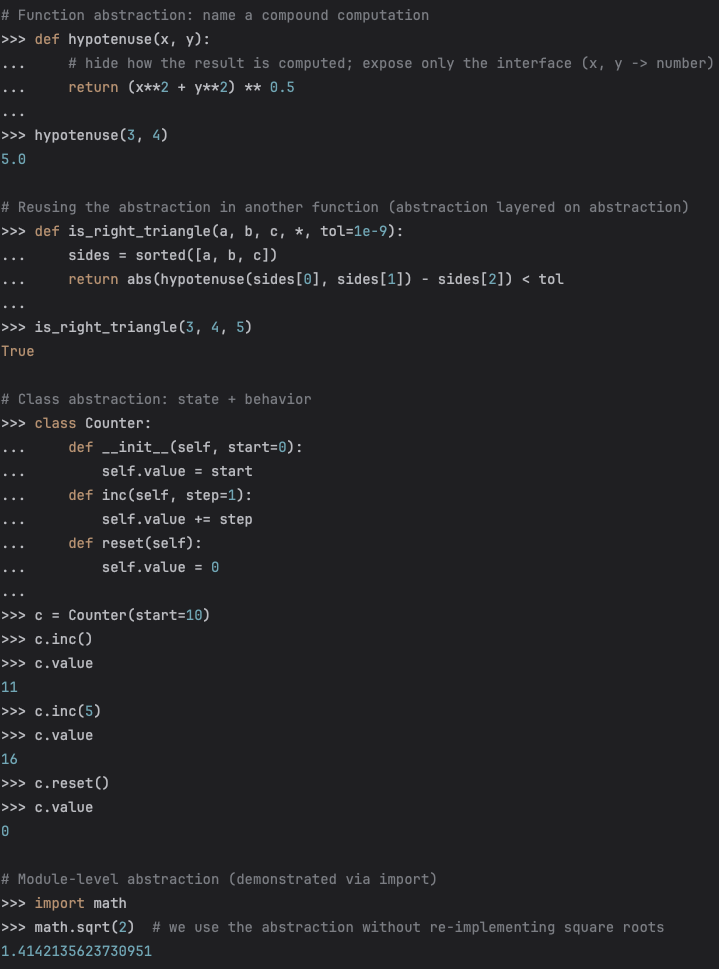


What this shows: We take small pieces (literals, names, simple calls), and combine them through operators, data structures, and control flow to express richer computations.

1. Means of abstraction Abstraction lets us give names to complex things and treat them as single units. This reduces repetition, clarifies intent, and makes programs easier to modify. In Python, key abstractions include:

* Variables: name a value so we can reuse it.
* Functions: package a computation behind a name and parameters.
* Classes and objects: bundle state (attributes) and behavior (methods).
* Modules: group related definitions into a file or package.

Interactive Python examples:



What this shows: We encapsulate details (the “how”) behind names and interfaces (the “what”). Functions hide computations, classes hide data representation and related operations, and modules hide entire libraries of functionality.

Putting it together

* Primitives give us the smallest building blocks.
* Combination lets us assemble these blocks into useful structures and behaviors.
* Abstraction lets us name and reuse those structures as single units, keeping programs clear and scalable.

Why this matters

* Without primitives, there’s nothing to build from.
* Without combination, we can’t express complex logic.
* Without abstraction, code becomes repetitive, fragile, and hard to understand.