



a named block that does one job, called by name whenever that job is needed again



Why functions

- a function packs up a block of code under one name, so it can be run again without retyping it
- it breaks a big program into smaller, named, testable pieces, each doing one clear job
- once written and checked, a function can be called as many times as needed, with different values each time

Three kinds of functions

- built-in functions come ready with Python, e.g. `print()`, `len()`, `range()`, no import needed
- module functions live inside a library and need an import first, e.g. `math.factorial()`, `random.randint()`
- user-defined functions are the ones written with `def`, built for one specific job
- all three are called the same way: a name followed by parentheses

Functions



one root, three sources a function can come from

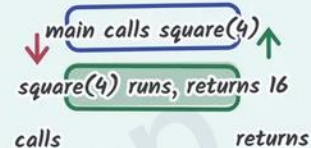
Defining and calling a function *

- a function is defined once, with `def name(parameters):` and a colon, then an indented body
- nothing inside the body runs when it is defined, only when the function is actually called
- calling it is just writing its name followed by parentheses, with any values it needs inside them
- the same function can be called many times, from many different places in the program

Defining and calling square

Q. `def square(n): return n * n`, then `answer = square(4)`

- 1 the `def` line only creates the function, it does not run the body yet
- 2 `square(4)` is the call, control jumps into the function with `n` bound to 4
- 3 `return n * n` computes 16 and sends it straight back to the call, ending the function there
- 4 `answer` stores the value handed back, 16



two frames, the arrow down the left marked calls carrying control into the function, the arrow up the right marked returns carrying the value straight back

Answer **answer is 16**

return sends a value back

- `return` sends a value to the caller and ends the function right there, nothing written after it runs
- a function with no return statement, or a bare `return`, still gives back a value: `None`
- `print()` inside a function only shows something on screen, it never hands a value back to the caller, mixing the two up is a common error

Handy math module functions

- `import math` is needed first, module functions are never available on their own
- `math`'s square-root function gives the square root of a number, `math.factorial(n)` gives `n` factorial, `math.pow(x, y)` gives `x` raised to `y` as a float
- `math.ceil(x)` rounds up to the next whole number, `math.floor(x)` rounds down

