Python Recursion

Table of contents

A **recursive function** is a function that calls itself. Every recursion needs at least one of each:

- **Base case**: a case defined outright, with no further recursive call (`if base_case: return constant`).
- **Recursive step**: a line where the function calls itself on "smaller" input, making progress toward a base case.

Canonical example: factorial

```
def fact(n: int) -> int:
    if not n:              # base case
        return 1
    return n * fact(n-1)   # recursive step
```

Evaluation has two phases:

- **Winding**: recursive calls build up a stack of pending computations (`4 × (3 × (2 × (1 × fact(0))))`).
- **Unwinding**: once the base case is hit, each pending computation resolves from the inside out.

If a base case is never reached, the recursion "bottoms out" with a `RecursionError: maximum recursion depth exceeded` once Python's call stack limit (1000 by default) is hit. This limit can be raised with `sys.setrecursionlimit(n)`, which is sometimes necessary for algorithms that legitimately need deeper recursion (as opposed to a genuine infinite recursion bug).

Choosing base cases

Type	Typical base case
`int` (counting down)	`0`
`list`	`[]`
`str`	`''` (empty string)

For less obvious base cases, consider the smallest example that still needs one recursive call (the *singleton* case), and work out what value the base case must return for that example to behave correctly.

Linear vs. tree recursion

Most recursive functions make a single recursive call per invocation (linear recursion, e.g. `fact`, `is_palindrome`). Some make more than one (tree recursion), e.g. checking whether any sublist sums to a target:

```
def sublist_sum(xs: list[int], target: int) -> bool:
    if not xs:
        return not target
    return sublist_sum(xs[1:], target-xs[0]) or sublist_sum(xs[1:], target)
```

Tree recursion can recompute the same subproblem many times. **Dynamic programming** — caching previous results — avoids this when subproblems overlap heavily (the classic example being naive recursive Fibonacci, which is exponential without a cache but linear with one).

Relationship to induction

Recursion is essentially the Principle of Mathematical Induction realised as code: a base case (the $P(0)$ case) plus a step that reduces a general case to a smaller one already known to work (the $P(n) \implies P(\text{succ}(n))$ step).

See python-scope[1] for how local variables/recursive calls interact with scope, and python-composition[2]/python-inheritance[3] for other structuring tools. See towers-of-hanoi[4] for a worked induction/recursion example.

[1] courses/csse1001/python-scope.pdf
[2] courses/csse1001/python-composition.pdf
[3] courses/csse1001/python-inheritance.pdf
[4] courses/csse1001/towers-of-hanoi.pdf