

← [course home \(/table-of-contents#section\\_javascript\\_concept\\_in-place\)](/table-of-contents#section_javascript_concept_in-place)

# In-Place Algorithm

An **in-place** function modifies data structures or objects outside of its own stack frame (i.e.: stored on the process heap or in the stack frame of a calling function). Because of this, the changes made by the function remain after the call completes.

In-place algorithms are sometimes called **destructive**, since the original input is "destroyed" (or modified) during the function call.

**Careful: "In-place" does *not* mean "without creating any additional variables!"** Rather, it means "without creating a new copy of the input." In *general*, an in-place function will only create additional variables that are  $O(1)$  space.

An **out-of-place** function doesn't make any changes that are visible to other functions. Usually, those functions copy any data structures or objects before manipulating and changing them.

In many languages, **primitive** values (integers, floating point numbers, or characters) are copied when passed as arguments, and more complex **data structures** (lists, heaps, or hash tables) are passed by reference. This is what Python does.

Here are two functions that do the same operation on a list, except one is in-place and the other is out-of-place:

```
def square_list_in_place(int_list):
    for index, element in enumerate(int_list):
        int_list[index] *= element

    # NOTE: no need to return anything - we modified
    # int_list in place

def square_list_out_of_place(int_list):
    # We allocate a new list with the length of the input list
    squared_list = [None] * len(int_list)

    for index, element in enumerate(int_list):
        squared_list[index] = element ** 2

    return squared_list
```

**Working in-place is a good way to save time and space.** An in-place algorithm avoids the cost of initializing or copying data structures, and it usually has an  $O(1)$  space cost.

**But be careful: an in-place algorithm can cause side effects.** Your input is "destroyed" or "altered," which can affect code *outside* of your function. For example:

```
original_list = [2, 3, 4, 5]
square_list_in_place(original_list)

print "original list: %s" % original_list
# Prints: original list: [4, 9, 16, 25], confusingly!
```

**Generally, out-of-place algorithms are considered safer because they avoid side effects.**

You should only use an in-place algorithm if you're space constrained or you're *positive* you don't need the original input anymore, even for debugging.

← [course home \(/table-of-contents\)](/table-of-contents)

Next up: JavaScript Scope ➡ (/question/js-scope?  
course=fc1&section=javascript)

---

Want more coding interview help?

Check out **interviewcake.com** for more advice, guides, and practice questions.