DATA C88C
Spring 2026

Mutability
Discussion 4: February 16, 2026

# Mutability

Some objects in Python, such as lists and dictionaries, are **mutable**, meaning that their contents or state can be changed. Other objects, such as numeric types, tuples, and strings, are **immutable**, meaning they cannot be changed once they are created.

There are many list mutation methods:

- `append(elem)`: Add `elem` to the end of the list. Return `None`.
- `extend(s)`: Add all elements of iterable `s` to the end of the list. Return `None`.
- `insert(i, elem)`: Insert `elem` at index `i`. If `i` is greater than or equal to the length of the list, then `elem` is inserted at the end. This does not replace any existing elements, but only adds the new element `elem`. Return `None`.
- `remove(elem)`: Remove the first occurrence of `elem` in list. Return `None`. Errors if `elem` is not in the list.
- `pop(i)`: Remove and return the element at index `i`.
- `pop()`: Remove and return the last element.

Dictionaries also have item assignment and `pop`.

```
>>> d = {2: 3, 4: 16}
>>> d[2] = 4
>>> d[3] = 9
>>> d
{2: 4, 4: 16, 3: 9}
>>> d.pop(4)
16
>>> d
{2: 4, 3: 9}
```

### Q1: Apply in Place

Implement `apply_in_place`, which takes a one-argument function `fn` and a list `s`. It modifies `s` so that each element is the result of applying `fn` to that element. It returns `None`.

```
def apply_in_place(fn, s):
    """Replace each element x of s with fn(x).

    >>> original_list = [5, -1, 2, 0]
    >>> apply_in_place(lambda x: x * x, original_list)
    >>> original_list
    [25, 1, 4, 0]
    """
    for i in range(len(s)):
        s[i] = fn(s[i])
```

One approach is to use `for i in range(...)` to iterate over the indices (positions) of `s`.

### Q2: Shuffle

Define a function `shuffle` that takes a sequence with an even number of elements (cards) and creates a new list that interleaves the elements of the first half with the elements of the second half.

To interleave two sequences `s0` and `s1` is to create a new sequence such that the new sequence contains (in this order) the first element of `s0`, the first element of `s1`, the second element of `s0`, the second element of `s1`, and so on.

> **Note:** If you're running into an issue where the special heart / diamond / spades / clubs symbols are erroring in the doctests, feel free to copy paste the below doctests into your file as these don't use the special characters and should not give an "illegal multibyte sequence" error.

```
def shuffle(s):
    """Return a shuffled list that interleaves the two halves of s.

    >>> shuffle(range(6))
    [0, 3, 1, 4, 2, 5]
    >>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
    >>> shuffle(letters)
    ['a', 'e', 'b', 'f', 'c', 'g', 'd', 'h']
    >>> shuffle(shuffle(letters))
    ['a', 'c', 'e', 'g', 'b', 'd', 'f', 'h']
    >>> letters  # Original list should not be modified
    ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
    """
    assert len(s) % 2 == 0, 'len(seq) must be even'
    half = len(s) // 2
    shuffled = []
    for i in range(half):
        shuffled.append(s[i])
        shuffled.append(s[half + i])
    return shuffled
```

### Q3: Happy Givers

In a certain discussion section, some people exchange gifts for the holiday season. We call two people **happy givers** if they give gifts to each other. Implement a function `happy_givers`, which takes in a `gifts` dictionary that maps people in the section to the person they gave a gift to. `happy_givers` outputs a list of all the happy givers in the section. The order of the list does not matter.

Note that if someone received but did not give a gift, they will not appear in the `gifts` dictionary as a key. (They'll appear only as a value.) You can assume that no one gives themself a gift.

Once you've found a solution, as a challenge, attempt to implement your solution in one line using a list comprehension.

```
def happy_givers(gifts):
    """
    >>> gift_recipients = {
    ...     "Alice": "Eve", # Alice gave a gift to Eve
    ...     "Bob": "Finn",
    ...     "Christina": "Alice",
    ...     "David": "Gina", # Gina is not a key because she didn't give anyone a gift
    ...     "Eve": "Alice",
    ...     "Finn": "Bob",
    ... }
    >>> list(sorted(happy_givers(gift_recipients))) # Order does not matter
    ['Alice', 'Bob', 'Eve', 'Finn']
    """
    output = []
    for giver in gifts:
        recipient = gifts[giver]
        if recipient in gifts and gifts[recipient] == giver:
            output = output + [giver]
    return output
    # Alternate solution using list comprehension
    # return [giver for giver in gifts if gifts[giver] in gifts and gifts[gifts[giver]]
    == giver]
```