

BINARY NUMBERS, CIRCUITS, AND MUTATION

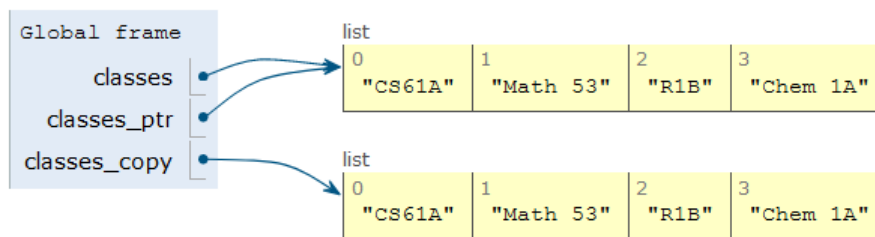
COMPUTER SCIENCE MENTORS

October 5 to October 8, 2020

1 Mutation

Lets imagine its your first year at Cal, and you have signed up for your first classes!

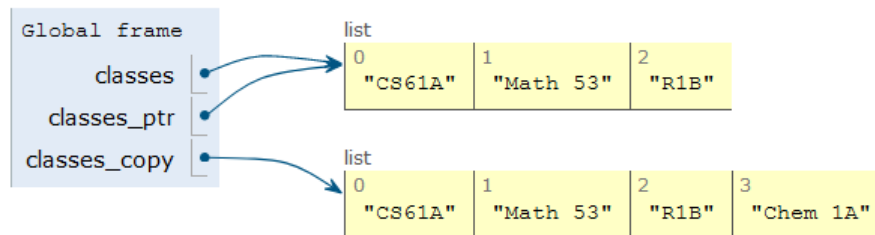
```
>>> classes = ["CS61A", "Math 53", "R1B", "Chem 1A"]
>>> classes_ptr = classes
>>> classes_copy = classes[:]
```



After a few weeks, you realize that you cannot keep up with the workload and you need to drop a class. Youve chosen to drop Chem 1A. Based on what we know so far, to change our classes list, we would have to create a new list with all the same elements as the original list except for Chem 1A. But that is silly, since all we really need to do is remove the Chem 1A element from our list.

We can fix this issue with list mutation. In Python, some objects, such as lists and dictionaries, are mutable, meaning that their contents or state can be changed over the course of program execution. Other objects, such as numeric types, tuples, and strings are immutable, meaning they cannot be changed once they are created. Therefore, instead of creating a new list, we can just call `classes.pop()`, which removes the last element from the list.

```
>>> classes.pop() # pop returns whatever item it removed
"Chem 1A"
```



List methods that mutate:

- `append(el)`: Adds `el` to the end of the list
- `extend(lst)`: Extends the list by concatenating it with `lst`
- `insert(i, el)`: Insert `el` at index `i` (does not replace element but adds a new one)
- `remove(el)`: Removes the first occurrence of `el` in list, otherwise errors
- `pop(i)`: Removes and returns the element at index `i`, if you do not include an index it pops the last element of the list

Ways to copy: list splicing (`[start:end:step]`), `list(...)`

1. What would Python display? If an error occurs, write "Error". If a function is displayed, write "Function". If nothing is returned, write "Nothing".

```
>>> a = [1, 2]
>>> b = a
>>> print (a.append([3, 4]))

>>> a

>>> b

>>> c = a[:]
>>> a[0] = 5
>>> a[2][0] = 6
>>> c

>>> a.extend([7, 8])
>>> a += [9]
>>> a += 10

>>> a

>>> print (c.pop(), c)
```

2. Given some list `lst`, possibly a deep list, mutate `lst` to have the accumulated sum of all elements so far in the list. If there is a nested list, mutate it to similarly reflect the accumulated sum of all elements so far in the nested list. Return the total sum of the original `lst`.

Hint: The **`isinstance`** function returns True for **`isinstance(l, list)`** if `l` is a list and False otherwise.

```
def accumulate(lst):
    """
    >>> l = [1, 5, 13, 4]
    >>> accumulate(l)
    23
    >>> l
    [1, 6, 19, 23]
    >>> deep_l = [3, 7, [2, 5, 6], 9]
    >>> accumulate(deep_l)
    32
    >>> deep_l
    [3, 10, [2, 7, 13], 32]
    """
    sum_so_far = 0
    for _____:
        _____
        if isinstance(_____, list):
            inside = _____
            _____
        else:
            _____
            _____
    return _____
```