# BINARY NUMBERS, MUTATION, AND REVIEW Solutions

COMPUTER SCIENCE MENTORS

October 5 to October 8, 2020

## 1 Binary Tables

1. Fill out the following table. Write N/A if the conversion is not possible. Some entries have already been filled out for you.

| Decimal | Binary (unsigned) | Binary (two's complement) |
|---|---|---|
| 25 | | |
| | 0010 1010 | |
| | | 0011 1100 |
| -66 | | |
| | | 1010 1000 |

- To convert from binary to decimal, write out which powers of two correspond to a 1, and sum them together.
- To convert from decimal to binary, it can be useful to create a binary place value table.

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

  – Starting with the closest power of two **smaller** than our decimal, set it to 1 in the table and subtract this from the decimal. Repeat until the decimal is zero.

- To convert from a negative decimal to two's complement binary, begin by finding the closest power of two **larger** than the decimal. For 66, this is 128.

| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

  – Set this value to 1 in the table and **add** this value to the decimal (since it is really a negative number) which should result in a positive decimal.

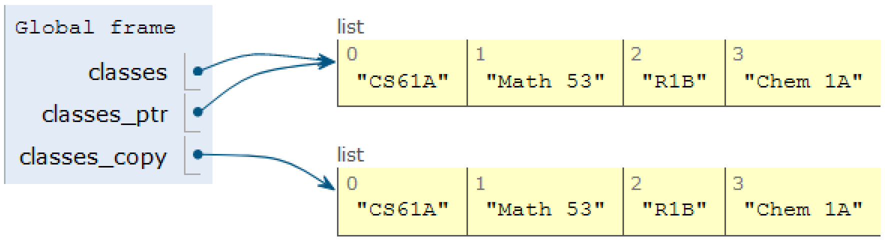
  – Then, we use normal binary conversion to shrink the decimal to zero.

| Decimal | Binary (unsigned) | Binary (two's complement) |
|---|---|---|
| 25 | 0001 1001 | 0001 1001 |
| 42 | 0010 1010 | 0010 1010 |
| 60 | 0011 1100 | 0011 1100 |
| -66 | N/A | 1011 1110 |
| -88 | N/A | 1010 1000 |

## 2 Mutation

Let's imagine it's your first year at Cal, and you have signed up for your first classes!

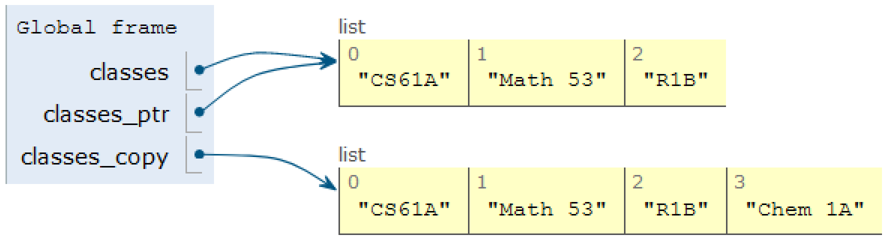
```
>>> classes = ["CS61A", "Math 53", "R1B", "Chem 1A"]
>>> classes_ptr = classes
>>> classes_copy = classes[:]
```



After a few weeks, you realize that you cannot keep up with the workload and you need to drop a class. You've 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]))
```

```
None
```

```
>>> a
```

```
[1, 2, [3, 4]]
```

```
>>> b
```

```
[1, 2, [3, 4]]
```

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

```
[1, 2, [6, 4]]
```

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

```
TypeError: 'int' object is not iterable
```

```
>>> a
```

```
[5, 2, [6, 4], 7, 8, 9]
```

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

[6, 4] [1, 2]
```

**Mutability in Lists**

| **Function** | **Create or Mutate** | **Action/Return Value** |
|---|---|---|
| lst.**append**(element) | mutate | attaches element to end of the list and returns None |
| lst.**extend**(iterable) | mutate | attaches each element in iterable to end of the list and returns None |
| lst.**pop**() | mutate | removes last element from the list and returns it |
| lst.**pop**(index) | mutate | removes element at index and returns it |
| lst.**remove**(element) | mutate | removes element from the list and returns None |
| lst.**insert**(index, element) | mutate | inserts element at index and pushes rest of elements down and returns None |
| lst += lst2 | mutates | attaches lst2 to the end of lst and returns None<br>same as lst.extend(lst2) |
| lst[start:end:step size] | create | creates a new list that start to stop (exclusive) with step size and returns it |
| lst = lst2 + [1, 2] | create | creates a new list with elements from lst2 and [1, 2] and returns it |
| **list**(iterable) | create | creates new list with elements of iterable and returns it |

(credits: Mihira Patel)

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 ____________________________________
```

```
def accumulate(lst):
    sum_so_far = 0
    for i in range(len(lst)):
        item = lst[i]
        if isinstance(item, list):
            inside = accumulate(item)
            sum_so_far += inside
        else:
            sum_so_far += item
            lst[i] = sum_so_far
    return sum_so_far
```

## 3 Review

1. Fill in `collapse`, which takes in a non-negative integer `n` and returns the number resulting from removing all digits that are equal to an adjacent digit, i.e. the number has no adjacent digits that are the same.

```
def collapse(n):
    """
    >>> collapse(12234441)
    12341
    >>> collapse(11200000013333)
    12013
    """
    rest, last = n // 10, n % 10

    if ___________________________________:

        ___________________________________

    elif _________________________________:

        ___________________________________
    else:

        ___________________________________
```

```
def collapse(n):
    rest, last = n // 10, n % 10
    if rest == 0:
        return last
    elif last == rest % 10:
        return collapse(rest)
    else:
        return collapse(rest) * 10 + last
```

2. Implement the function `make_change`, which takes in a non-negative integer amount in cents `n` and returns the minimum number of coins needed to make change for `n` using 1-cent, 3-cent, and 4-cent coins.

```
def make_change(n):
    """
    >>> make_change(5) # 5 = 4 + 1 (not 3 + 1 + 1)
    2
    >>> make_change(6) # 6 = 3 + 3 (not 4 + 1 + 1)
    2
    """

    if _____________________:
        return 0

    elif ___________________:

        ___________________________________

    elif ___________________:

        ___________________________________
    else:

        ___________________________________
```

```
def make_change(n):
    if n == 0:
        return 0
    elif n < 3:
        return 1 + make_change(n - 1)
    elif n < 4:
        return 1 + min(make_change(n - 1), make_change(n - 3))
    else:
        return 1 + min(make_change(n - 1), make_change(n - 3),
            make_change(n - 4))
```

3. Given a list of integers `lst`, return the maximum sum of a subset of size `n`. If `n` is greater than or equal to the length of `lst`, just return the sum of the elements `lst`.

```
def max_subset_sum(lst, n):
    """
    >>> max_subset_sum([1, 2, 3, 4], 2)
    7
    >>> max_subset_sum([1, 4, 2, 0, 6], 3)
    12
    """

    if _________________________:

        __________________________

    elif ________________________:

        __________________________

    with_elem = _____________________________ + _________

    without_elem = ____________________________________

    return ____________________________________________
```

```
def max_subset_sum(lst, n):
    if n == 0:
        return 0
    elif len(lst) <= n:
        return sum(lst)
    with_elem = max_subset_sum(lst[1:], n - 1) + lst[0]
    without_elem = max_subset_sum(lst[1:], n)
    return max(with_elem, without_elem)
```