```
## For example, "Mr. Justin Case" or "Miss Alice Inwonderland".
##
## Write a function new_title which consumes a name
## and produces a new name, which is the same as the first, except
## that if the title was "Miss" or "Mrs.", the new name has
## the title "Ms.".
##

# Problem 4:
## Write a Python function check_password which consumes a string and a
## natural number. The function then repeatedly asks the user to input the
## password, until it matches the consumed string or until the user has ma
## N incorrect guesses, where N is the consumed number. The function print
## Welcome
## if the password is guessed correctly and it produces True.
## If the function is not guessed correctly and the user runs out of guess
## then
## Access Denied
## is printed, and False produced.

# Problem 5
## Write a Python function draw_triangle that consumes a natural number, n
## and prints a triangle over n lines, as shown below. The function produces
## None
## draw_triangle(3) prints
## X
## XX
## XXX
##
## draw_triangle(6) prints
## X
## XX
## XXX
## XXXX
## XXXXX
## XXXXXX
```

Consider a new type: `Table`. A `Table` is a (listof (listof Int)), which is nonempty, and in which each list corresponds to a row of a `Table`. It is assumed that each row is nonempty and each row has the same number of entries as every other row.

For example, the following are tables:

```
t0 = [[1]]
t1 = [[1,2,3]]
t2 = [[1],[2],[3]]
t3 = [[1,2],[3,4],[5,6],[7,8]]
t4 = [[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16], [17,18,19,20]]
```

Write a function `sum_columns` that consumes a `Table t`, and produces a list containing the columns sums for `t`.

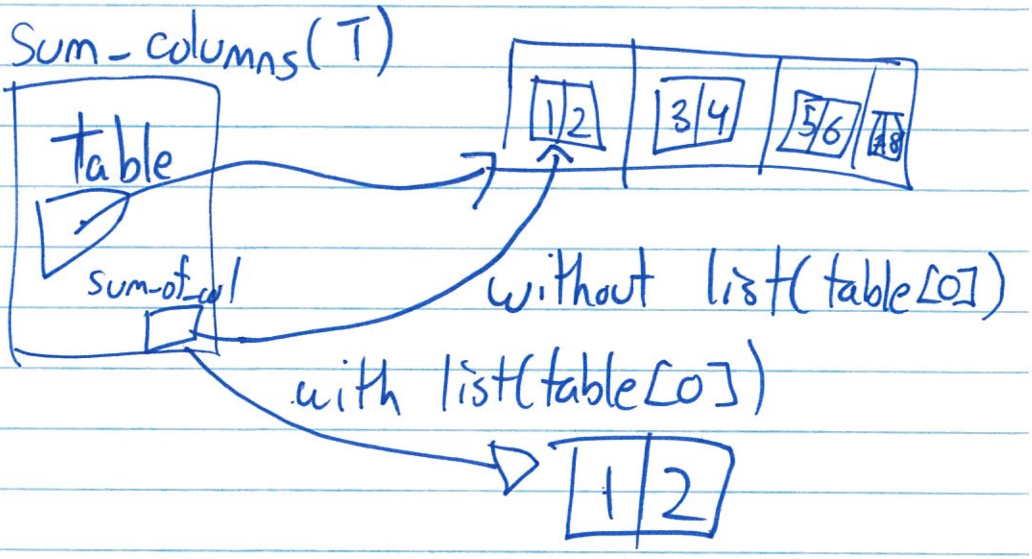
For example,

```
sum_columns(t0) => [1]
sum_columns(t1) => [1,2,3]
sum_columns(t3) => [16, 20]
```

Table.

1	2	3
4	5	6
7	8	9

~~1~~ ~~5~~ 12	~~2~~ ~~7~~ 15	~~3~~ ~~9~~ 18



```
# Problem 3:

# Use accumulative recursion to write the function spread
# that consumes a list of numbers, and produces the difference
# between the largest and smallest values in the list.
# For example, spread([3,1,9,17,-4,2]) => 21,
# spread([2]) => 0, spread([]) => 0.
```

```
#
# Problem 4:
# Use accumulative recursion to write a function majority that
# consumes a list of booleans and determines if there are more
# True than False values in the list.
# For example, majority([True, False, False]) => False,
# majority([False, True, False, True]) => False,
# majority([False, True, True, True, True]) => True
#
```

Spread.

3	1	9	17	-4	2

min = ~~3~~ ~~1~~ -4

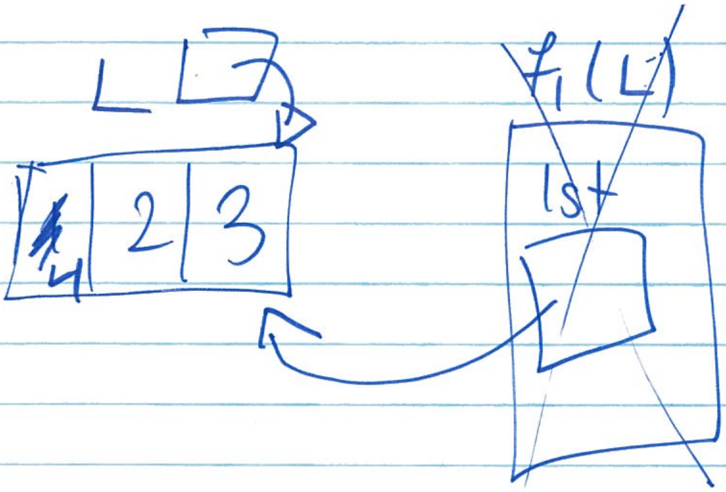
max = ~~3~~ ~~9~~ 17

acc = ~~0~~ ~~28~~ ~~16~~ 21

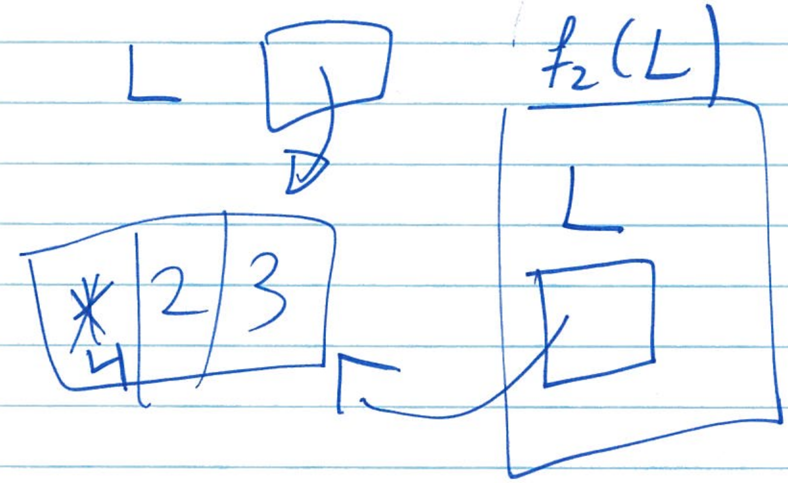
T F F

num-true ~~0~~ 1
num-falses 0 ~~1~~ 2

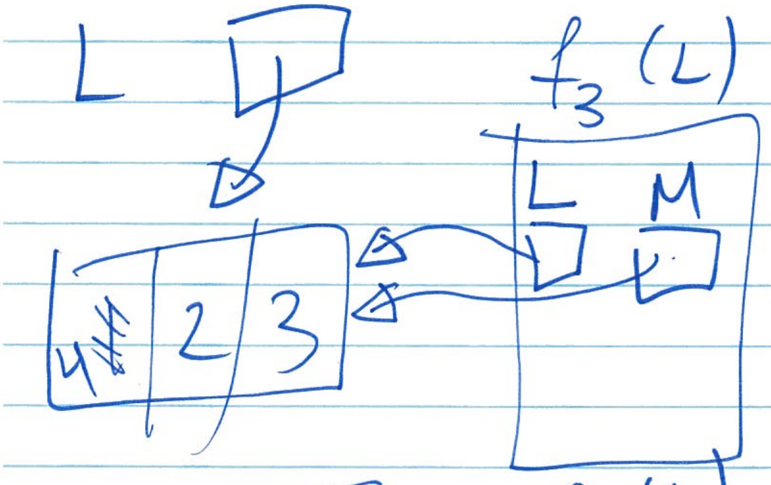
Base Case: If empty list:
~~If nu~~ return num_true >num_false



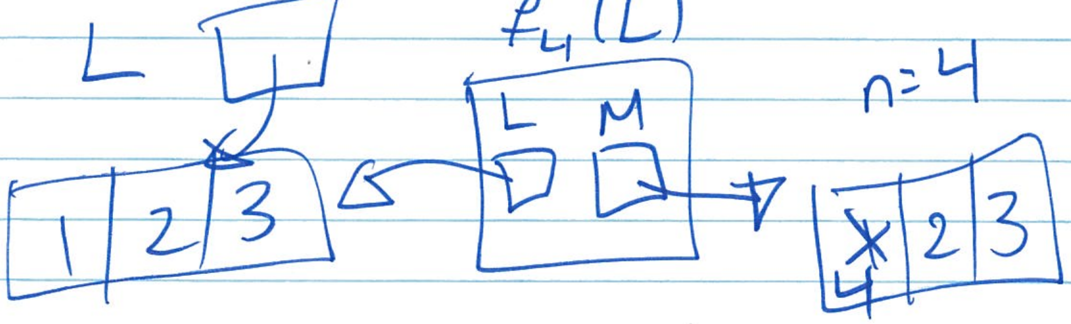
n=1



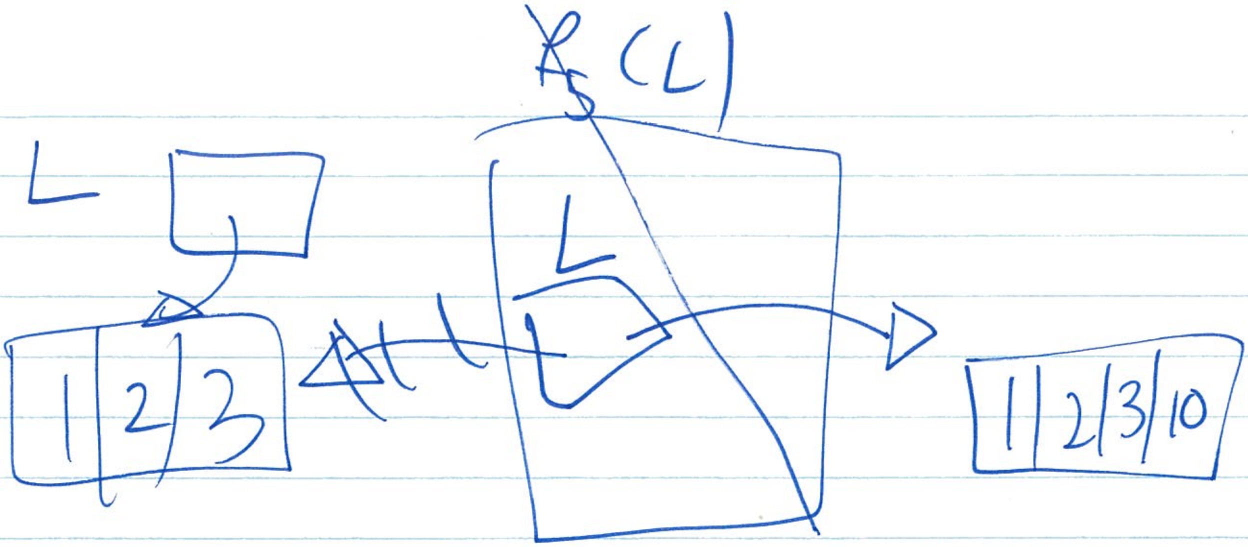
n=2.

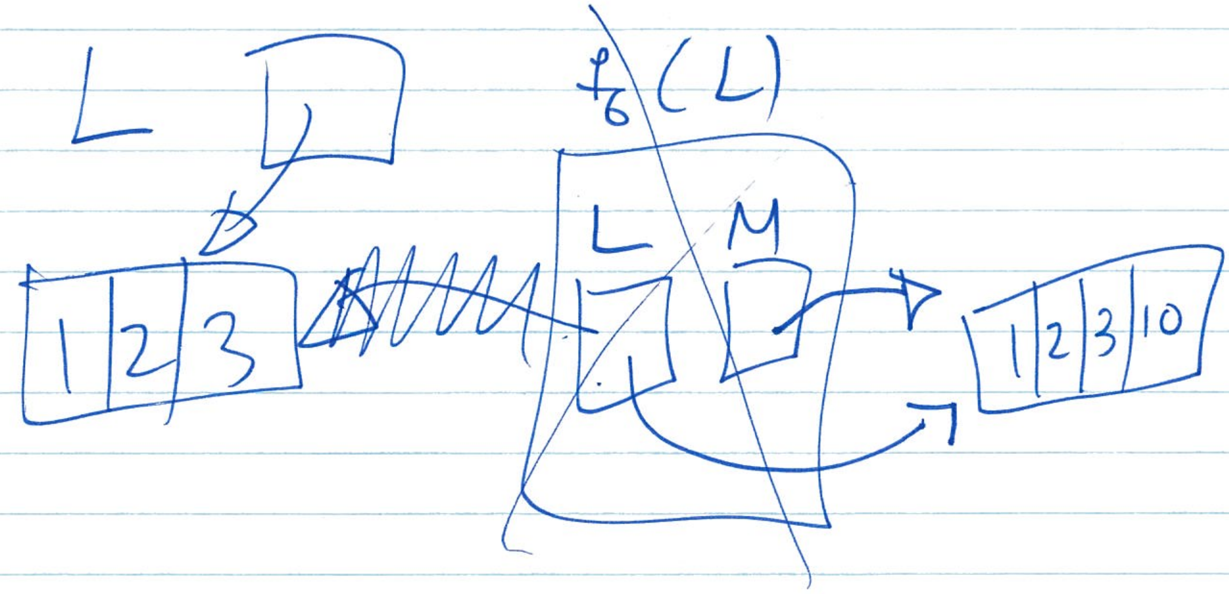


n=3



n=4


f5(L)
L
L
1 2 3
1 2 3 10


f6(L)
L
L
M
1 2 3
1 2 3 10

Write a function selective_add_one which consumes a list of integers L and produces a new list M consisting of all the integers of L greater than or equal to 7 except each value is also incremented by 1.