# Administrivia

- Start Homework earlier

# Lists

# Lists: Type of problems

- **Cat 1**: given a list L, process L and compute some value out of its elements. Example: `alternatingSum`, `hasDuplicates`
- **Cat 2:** Given some input, return a **NEW** list with some properties. Example: return the *first n 7ish* numbers, `findDuplicates`

# Example: alternating sum

- Given a list of positive integers, calculate the alternating sum
- `alternatingSum([1, 2, 3, 4]) == -2`
  - because 1 - 2 + 3 - 4 = -2
- `alternatingSum([15, 5, 10, 1]) == -2`
  - because 15 - 5 + 10 - 1 = 19

# Example: find duplicates

- Write the function `findDuplicates(L)` that takes a list `L` of arbitrary values, and returns the list of duplicate values.
- The order is not important, but there shouldn't be duplicate values in the result
- `assert(findDuplicates([1,1,1,2]) == [1])`
- `assert(findDuplicates([1,4,2,1,3,4,5,2]) ==[1,4,2])`

# What's really happening here?

```
L = [15112, 42, 15122, 2023]
L.append(2024)
```

`L = [15112, 42, 15122, 2023]`

```
def ....(theList):
    ...
```

`theList == [15112, 42, 15122, 2023]`

# Aliasing

- Using lists as function arguments
- [Python tutor: example]

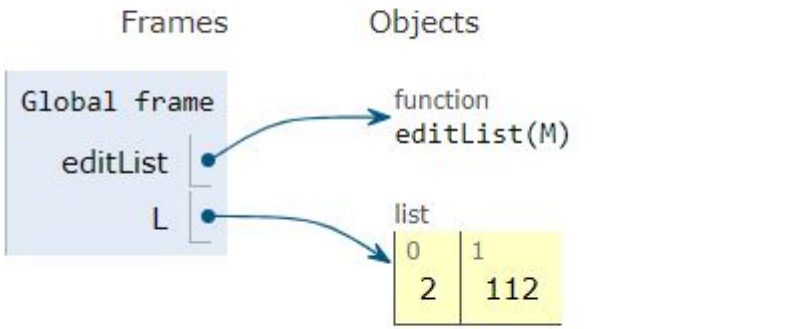
Python 3.6
known limitations

```
1  def editList(M):
2      M[0] =2
3
4
5  L = [15, 112]
6
7  editList(L)
8
9  print(L)
```

Edit this code

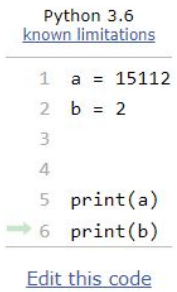
Print output (drag lower right corner to resize)

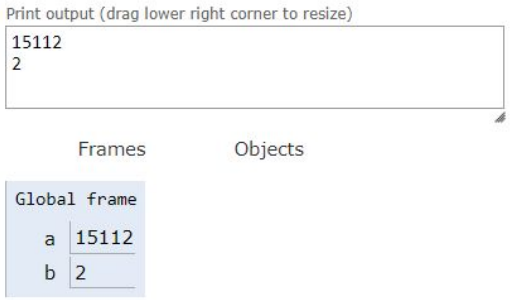
```
[2, 112]
```



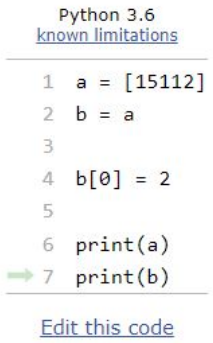
# Aliasing

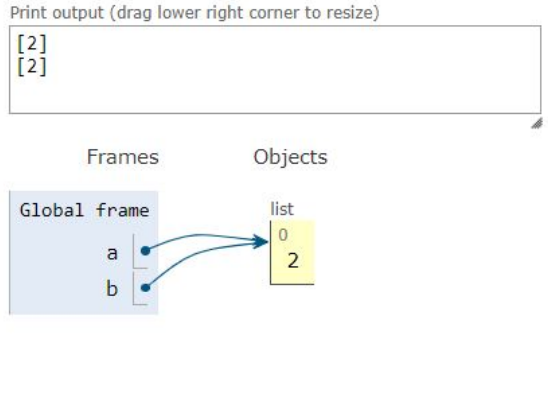
- Python tutor: example 1




- Python tutor: example 2



# Lists: Taxonomy of problems

- **Cat 1**: given a list L, process L and compute some value out of its elements. Example: `alternatingSum`, `hasDuplicates`
- **Cat 2:** Given some input, return a **NEW** list with some properti
  Example: return the *first n 7ish* numbers, `findDuplicates`

- **Cat 3:** Given a list L, write a ***non-destructive function*** to solve some task using L **without modifying** the original L.
  Example: `nondestructiveRemoveNonInts`
- **Cat 4:** Given a list L, write a ***destructive function*** to solve some task that **mutates** `L`.
  Example: `destructiveRemoveNonInts`

# Example: nondestructiveRemoveNonInts

- Write the function nondestructiveRemoveNonInts(L) that returns a new list that is equivalent to L without the non-integer values. It should not modify the original L.

```
L = [112,‘hi’,‘3’,1,5]
nondestructiveRemoveNonInts(L) returns [112,1,5]
assert(L == [112,‘hi’,‘3’,1,5])
```

# Example: `destructiveRemoveNonInts`

- Write the function `destructiveRemoveNonInts(L)` that removes the non-integer values from L.

```
L = [112,‘hi’,‘3’,1,5]
destructiveRemoveNonInts(L) # assumed to return None
assert(L == [112,1,5])
```

# Destructive vs Non-Destructive Operations

How do we add a value to a list **destructively**? Use append, insert, or +=.

```
lst = [1, 2, 3]
lst.append(5)
lst.insert(1, "foo")
lst += [10, 20] # Annoyingly different from lst = lst + [10, 20]
```

How do we add a value to a list **non-destructively**? Use variable assignment with list concatenation.

```
lst = [1, 2, 3]
lst = lst + [5, 10, 20]
```

# Destructive vs Non-Destructive Operations

How do we remove a value from a list **destructively**? Use remove or pop.

```
lst = [1, 2, 3]
lst.remove(2)  # remove the value 2
lst.pop(1)     # remove the value at index 1
```

How do we remove a value from a list **non-destructively**? Use variable assignment with list slicing.

```
lst = [1, 2, 3]
lst = lst[:1]
```

# Exercise

- At the end of this set of operations, which pairs of lists will still be aliased?

```
1 a = [ 1, 2, "x", "y" ]
2 b = a
3 c = [ 1, 2, "x", "y" ]
4 d = c
5 a.pop(2)
6 b = b + [ "wow" ]
7 c[0] = 42
8 d.insert(3, "yey")
```

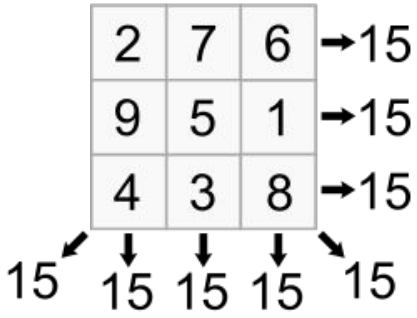
1. **a** and **b**
2. **c** and **d**
3. **All** lists are aliases of the **same** list
4. **None** of them are aliased

# Lists: Taxonomy of problems

- **Cat 1**: given a list L, process L and compute some value out of its elements. Example: `alternatingSum`, `hasDuplicates`
- **Cat 2:** Given some input, return a **NEW** list with some properties. Example: return the *first n 7ish* numbers, `findDuplicates`
- **Cat 3:** Given a list L, write a ***non-destructive function*** to solve some task using L **without modifying** the original L.
  - Example: `nondestructiveRemoveNonInts`
- **Cat 4:** Given a list L, write a ***destructive function*** to solve some task that **mutates** L**.**
  - Example: `destructiveRemoveNonInts`
- **Cat 5:** Solve a task involving **2d lists.**
  - Example: is`MagicSquare`

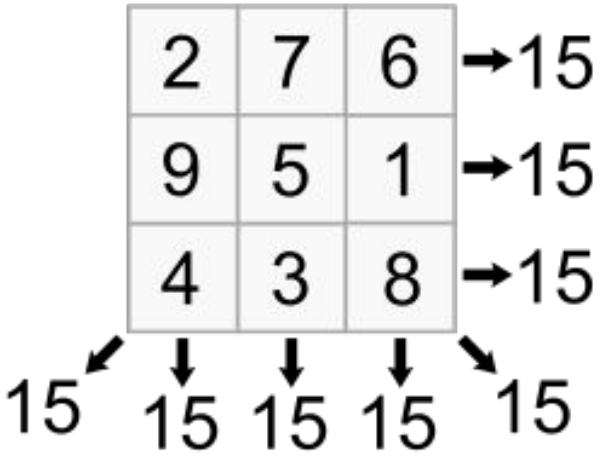
# isMagicSquare

- Write the function `isMagicSquare(L)` that takes an arbitrary list L and returns `True` if it is a magic square and False otherwise, where a magic square has these properties:
  - The list is 2d, non-empty, **square**, and contains **only integers**, where no integer occurs more than once in the square.
  - Each row, each column, and each of the 2 diagonals each sum to the same total. Note that we do not require that the integers are strictly in the range from 1 to n for some n. We only require that the integers are unique and that the sums are identical.



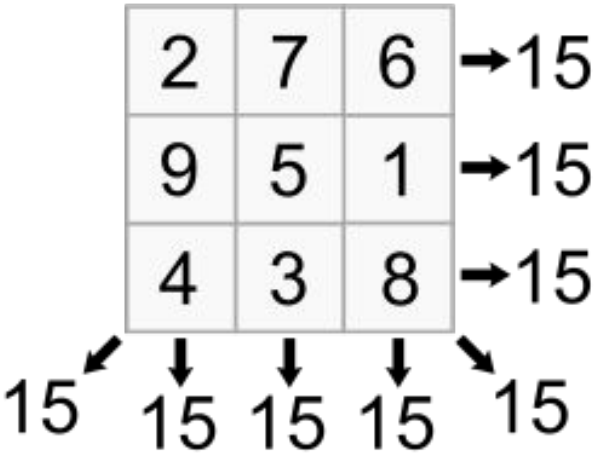

# `isMagicSquare(L)`

- Check that L is a list
- Check that L is a non-empty *square* 2d list
  - Check that all elements are lists
  - Check that all elements have `len(L)` elements
- Check that all elements in every row of L are integers
- Check that all elements in every row are unique
- Check that the sum of rows, the sum of columns, and the sum of diagonals are equal



# `isMagicSquare(L)`

- Check that L is a list
- Check that L is a non-empty *square* 2d list:
  - Check that all elements are lists
  - Check that all elements have `len(L)` elements
- Check that all elements in every row of L are integers
- Check that all elements in every row are unique
- Check that the sum of rows, the sum of columns, and the sum of diagonals are equal

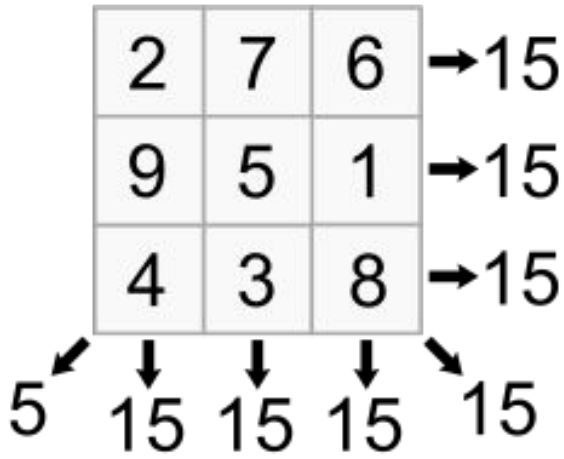


# isValidSquare(L)

isValidSquare(L)

- Check that L is a list
- Check that L is a non-empty *square* 2d list:
  - Check that all elements are lists
  - Check that all elements have `len(L)` elements
- Check that all elements in every row of L are integers
- Check that all elements in every row are unique

```
1  def isValidSquare(L):
2      if not isinstance(L, list):
3          return False
4      if len(L) == 0:
5          return False
6      for row in L:
7          if not isinstance(row, list):
8              return False
9      nrows = len(L)
10     for row in L:
11         if len(row) != nrows:
12             return False
13     for row in L:
14         for val in row:
15             if not isinstance(val, int):
16                 return False
17     valuesSeen = []
18     for row in L:
19         for val in row:
20             if val in valuesSeen:
21                 return False
22             valuesSeen.append(val)
23     return True
```



# isMagicSquare(L)

- Check that L is a list
- Check that L is a non-empty *square* 2d list:
  - Check that all elements are lists
  - Check that all elements have `len(L)` elements
- Check that all elements in every row of L are integers
- Check that all elements in every row are unique
- Check that the sum of rows, the sum of columns, and the sum of diagonals are equal