Lists, Aliasing, Mutability

Kelly Rivers and Stephanie Rosenthal

15-110 Fall 2019

Announcements

- Homework 3 Check-in 2 is due Monday at 12-noon!

  Start Early!
- Homework 3 full is also due in 2 Mondays.

Learning Objectives

- To define mutable and alias
- To distinguish mutable types from immutable types.
- To trace the values of variables by understanding their mutability and aliases
- To create 2D lists and iterate through them

Lists

Lists represent sequences, in order, of data

We can define, add, remove, combine, edit, iterate over lists.

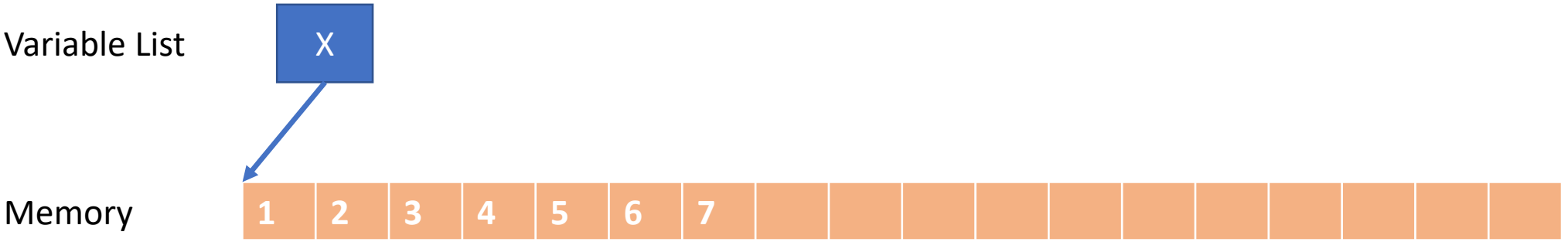
```
X = [1,2,3,4,5,6,7]
```

What is happening in memory? How can we keep adding/removing?

Lists and Memory

```
X = [1,2,3,4,5,6,7]
```

A variable points to a large place in memory designated for the list

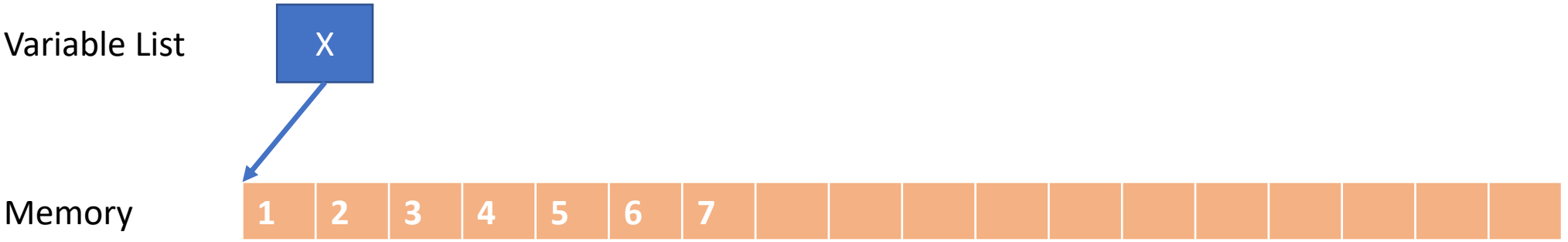


Lists and Memory

```
X = [1,2,3,4,5,6,7]
```

A variable points to a large place in memory designated for the list

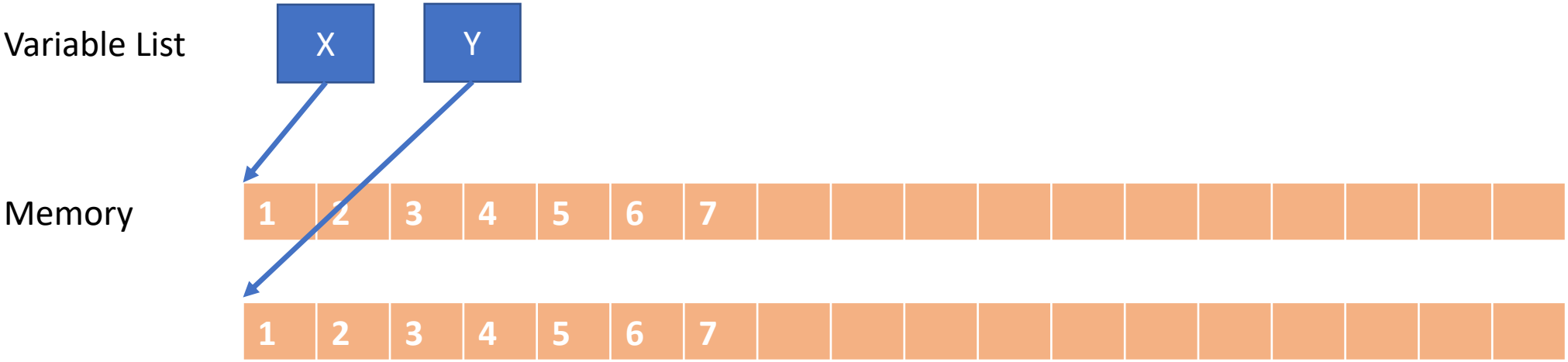
This space allows it to add and remove values without running out



Lists and Memory

```
X = [1,2,3,4,5,6,7]
Y = [1,2,3,4,5,6,7] #new list, new memory
```

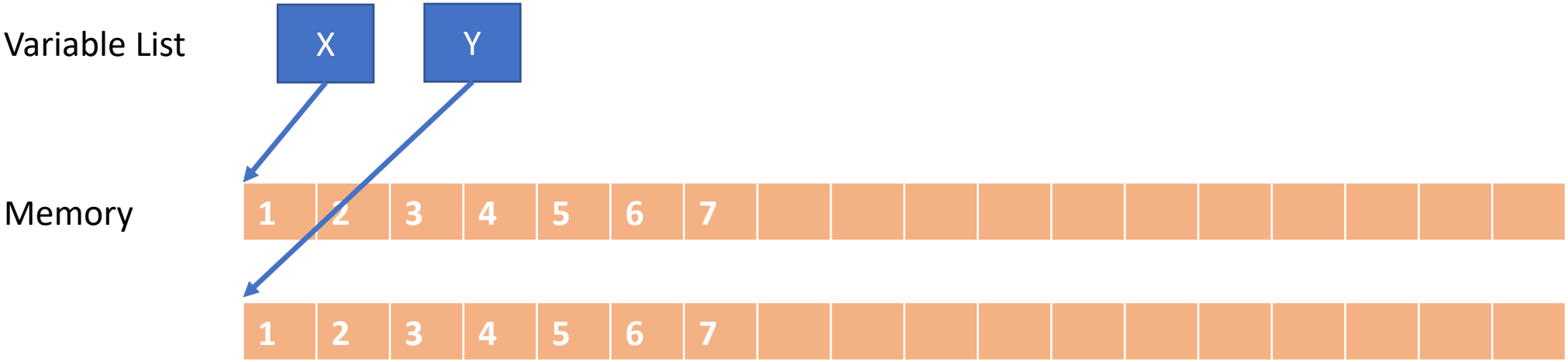
Python assumes you'll want to edit them separately



Lists and Memory

```
X = [1,2,3,4,5,6,7]
Y = X+[]                    #new list, new memory
```

Python assumes you'll want to edit them separately



Aliasing

```
X = [1,2,3,4,5,6,7]
Y = X
```

Setting another variable equal just points to the same address in memory

Modifying Lists with Aliasing

We can directly add or remove an item to/from a list!

```
L = [1,2,3,4,5,6,7,8,9]

L.append(10)    #10 gets inserted at the end of the list
L.remove(5)     #5 gets removed from the list
L.remove(5)     #nothing happens, there is not a second 5
L.insert(4,5)   #insert 5 at index 4 of the list
```

List Behavior with Aliasing vs Separate Copy

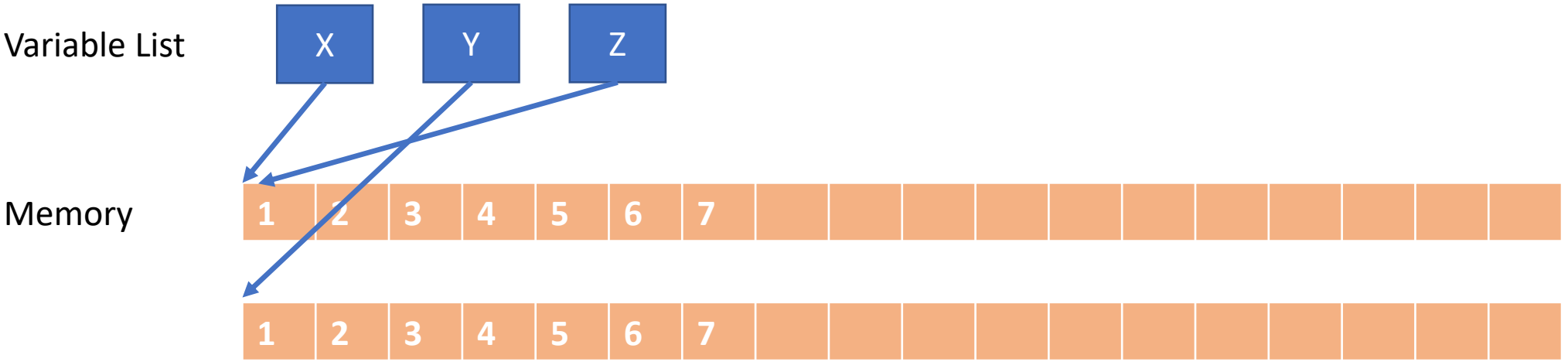
```
X = [1,2,3,4,5,6,7]
Y = [1,2,3,4,5,6,7] #new list, new memory
Z = X


X.remove(3)
print(X)
print(Y)
print(Z)
```

What will the lists look like?

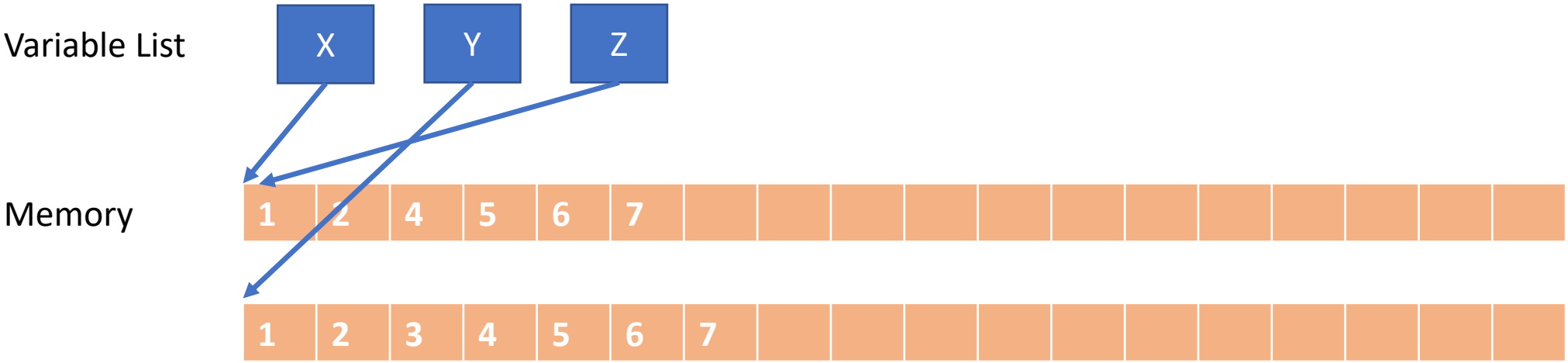
List Behavior with Aliasing vs Separate Copy

```
X = [1,2,3,4,5,6,7]
Y = [1,2,3,4,5,6,7] #new list, new memory
Z = X
```



List Behavior with Aliasing vs Separate Copy

```
X = [1,2,3,4,5,6,7]
Y = [1,2,3,4,5,6,7] #new list, new memory
Z = X
X.remove(3)
```



Checking Memory Locations

`id(X)` will return the memory location of the variable X

If `id(X) == id(Y)` then the variables are aliased

Otherwise, the values of the data are in separate memory locations

Aliasing Takeaways

Be careful if you are using lists and creating aliases vs copies

You may expect a list to change and it doesn’t or vice versa

Use the one that is correct for your algorithm/application

Why are copies different than aliases?

Lists are mutable – you can change the value of a list without changing its memory location

Insert, Append, and Remove all change the list in its location

Aliases point to the same location in memory, so they edit the same list.

Copies point to different locations in memory, so the lists are not the same.

Why are copies different than aliases?

Lists are mutable – you can change the value of a list without changing its memory location

Insert, Append, and Remove all change the list in its location

Aliases point to the same location in memory, so they edit the same list.

Copies point to different locations in memory, so the lists are not the same.

Not all data types are mutable. The opposite of mutable is immutable.

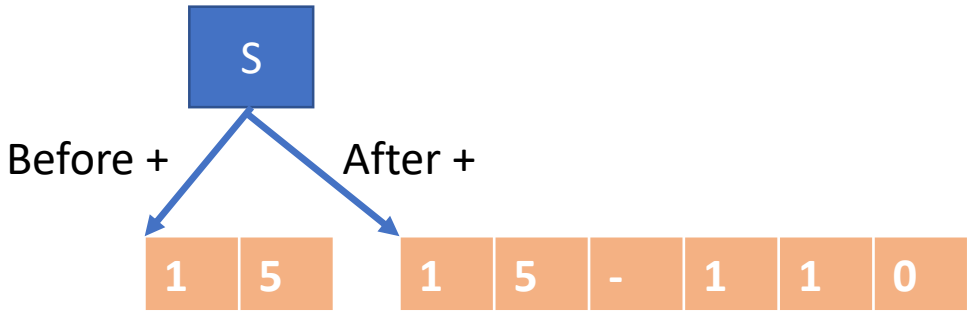
Are Strings Mutable?

```
S = "15"
print(id(S))
S = S+"-110"
print(id(S))
```

Do they print the same value?

Are Strings Mutable?

```
S = "15"
print(id(S))
S = S+"-110"
print(id(S))
```



Strings are immutable.

Each string is stored in a separate location in memory.

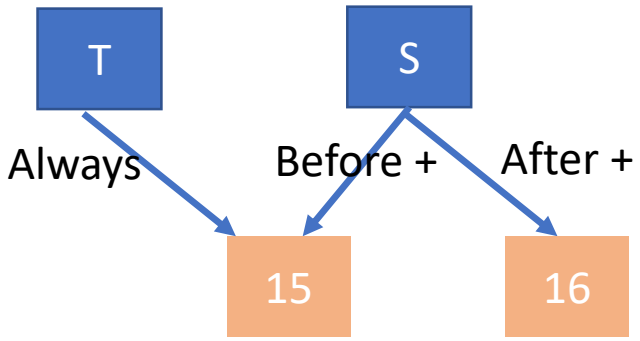
Are Integers and Floats Mutable?

```
S = 15
print(id(S))
T = S
S = S+1
print(id(S))
print(id(T))
print(S,T)
```

Do they have the same id? Do they have the same value?

Are Integers and Floats Mutable?

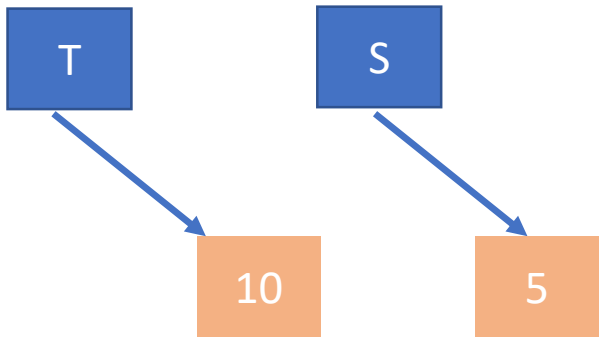
```
S = 15
print(id(S))
T = S
S = S+1
print(id(S))
print(id(T))
print(S,T)
```



Numbers are immutable. There is a separate location in the program for each.

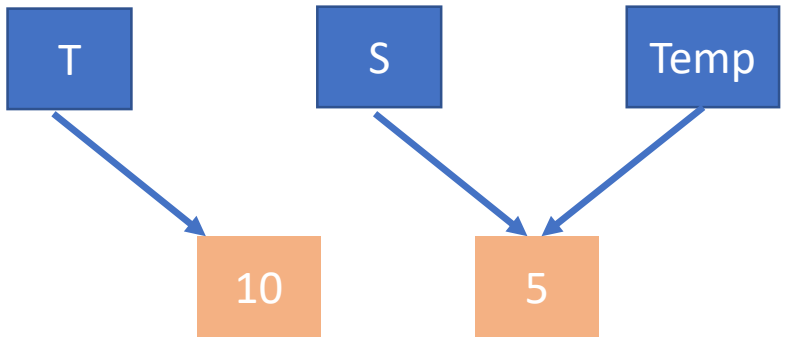
Swapping Variable Values

```
S = 5
T = 10
Temp = S
S = T
T = Temp
```



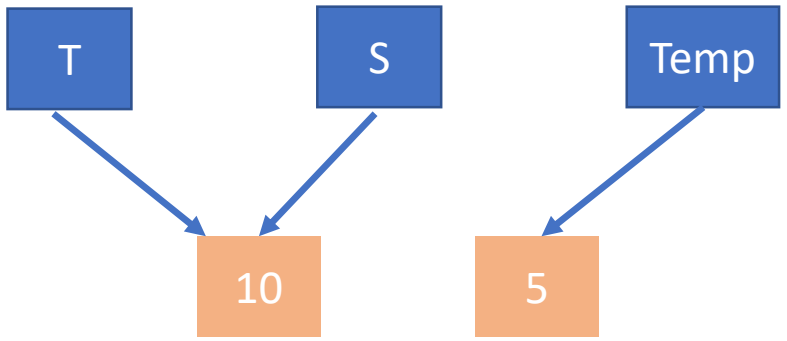
Swapping Variable Values

```
S = 5
T = 10
Temp = S
S = T
T = Temp
```



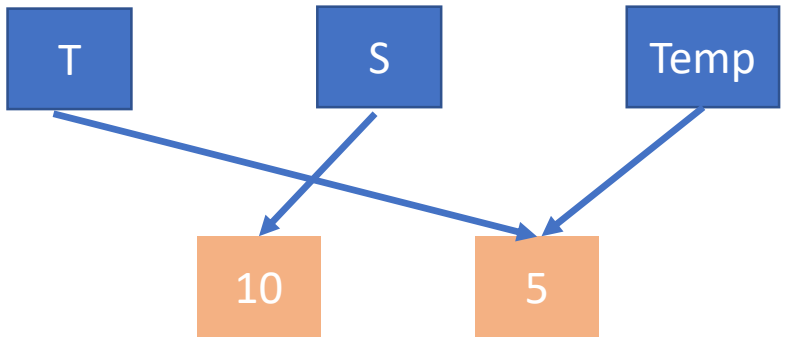
Swapping Variable Values

```
S = 5
T = 10
Temp = S
S = T
T = Temp
```



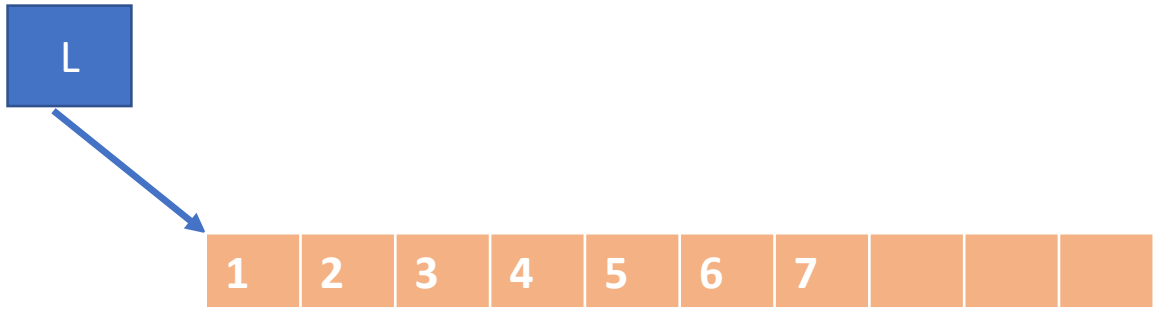
Swapping Variable Values

```
S = 5
T = 10
Temp = S
S = T
T = Temp
```



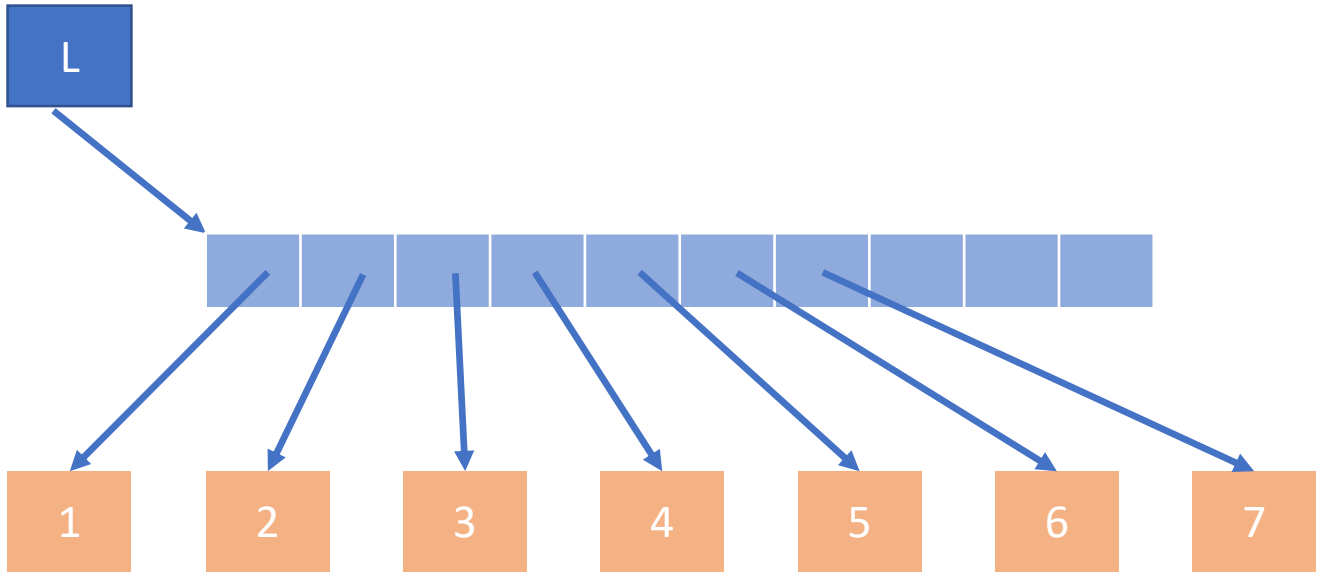
Swapping List Values

```
L = [1,2,3,4,5,6,7]
Temp = L[1]
L[1] = L[2]
L[2] = Temp
```



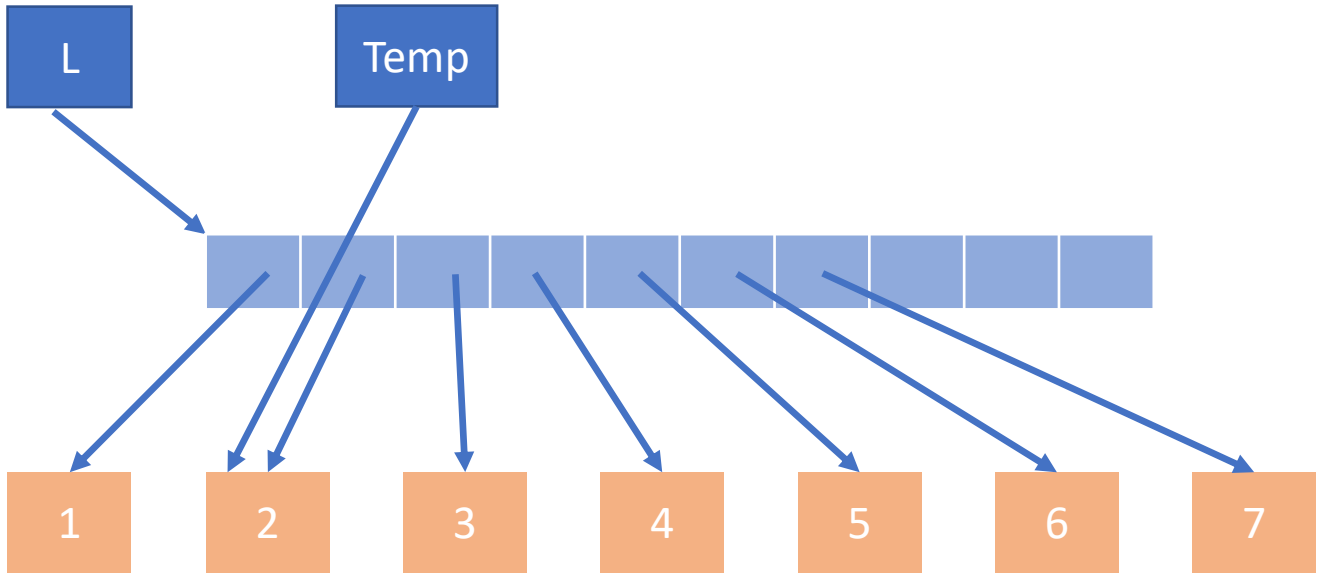
Swapping List Values

```
L = [1,2,3,4,5,6,7]
Temp = L[1]
L[1] = L[2]
L[2] = Temp
```



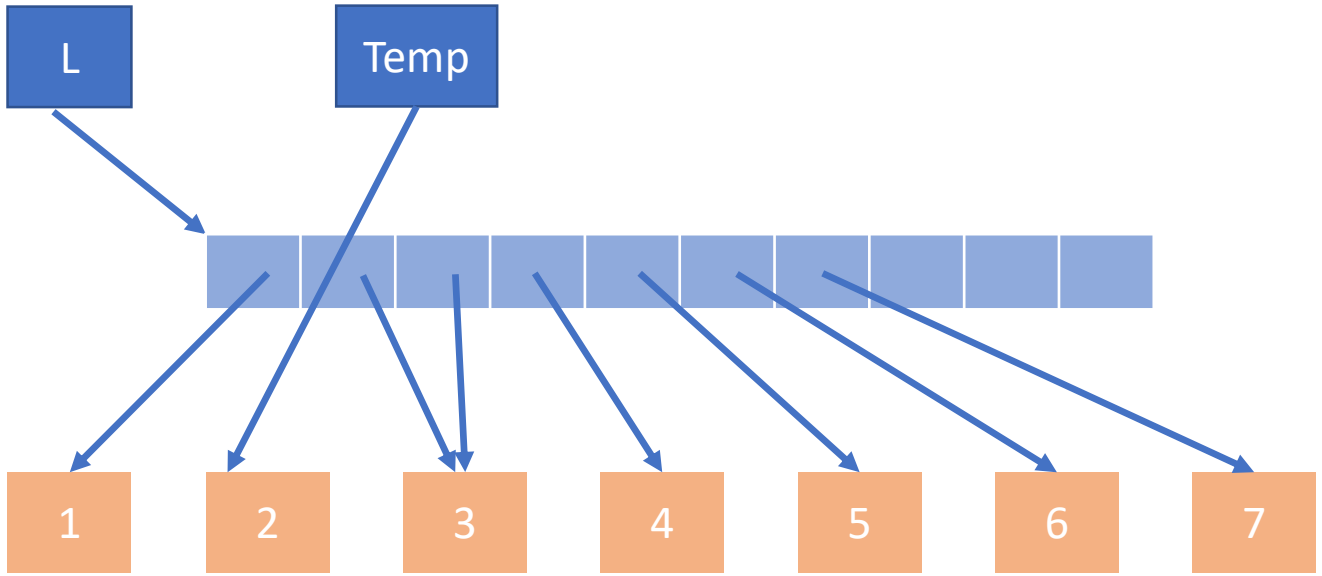
Swapping List Values

```
L = [1,2,3,4,5,6,7]
Temp = L[1]
L[1] = L[2]
L[2] = Temp
```



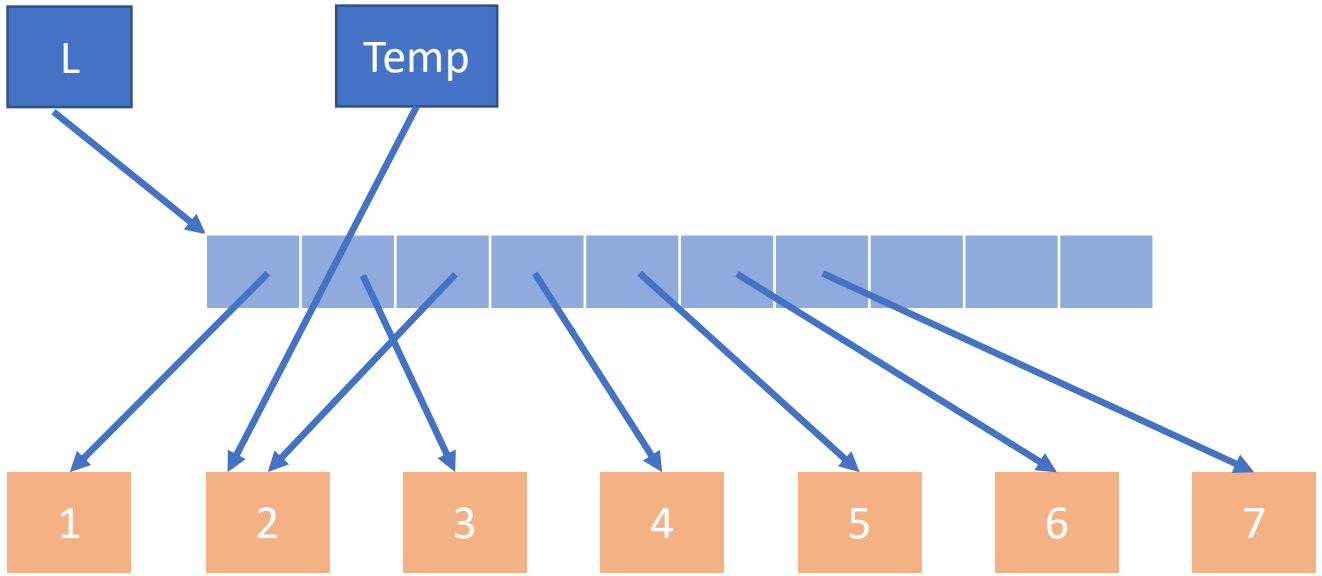
Swapping List Values

```
L = [1,2,3,4,5,6,7]
Temp = L[1]
L[1] = L[2]
L[2] = Temp
```



Swapping List Values

```
L = [1,2,3,4,5,6,7]
Temp = L[1]
L[1] = L[2]
L[2] = Temp
```

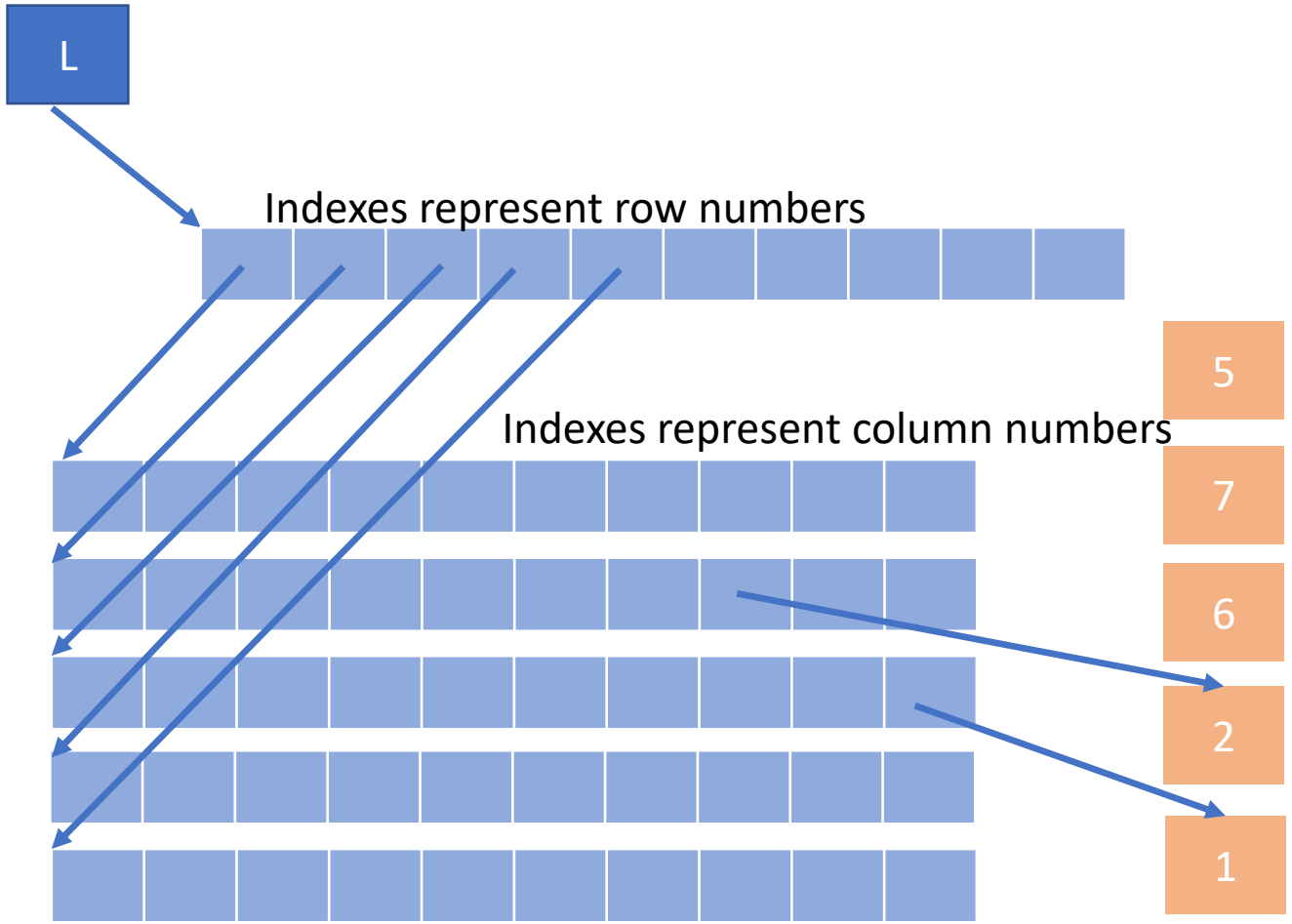


2D Lists

Now that we understand what's going on, it's easier to understand different ways that we can use lists.

Example:

We can make lists of lists



Iterating through 2D Lists

```
for i in L:
    for j in i:
        print(j)
```

```
for i in range(len(L)):
    for j in range(len(L[i])):
        print(L[i][j])
```

Note:
L[i] is a list
L[i][j] is an index in L[i]'s list