Algorithms and Data Structures for Data Science

Stacks and Queues and (maybe) 2D Lists

My white whale

CS 277
Brad Solomon

February 14, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

The purpose of assessments in CS 277

Reminder: MP 0 due today!

Informal Early Feedback

An anonymous survey about the class

If 70% of class completes, everyone gets bonus points

Please provide constructive criticism and positive feedback

Learning Objectives

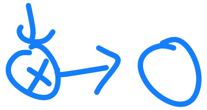
Observe data structure tradeoffs between data access and speed

Understand the fundamentals of the stack and queue

Introduce NumPy and practice 2D lists

Array Implementation

Given node in LL



	Singly Linked List	Array
Look up **arbitrary** location	O(n)	**O(1)**
Insert after **given** element	**O(1)**	O(n)
Remove after **given** element	**O(1)**	O(n)
Insert at **arbitrary** location	O(n)	O(n)
Remove at **arbitrary** location	O(n)	O(n)
Search for an input **value**	O(n)	O(n)

find ☹

Thinking critically about lists: tradeoffs

Can we make our lists better at some things? What is the cost?

Imagine I want to do a very complex calculation.

↳ $O(n^3)$

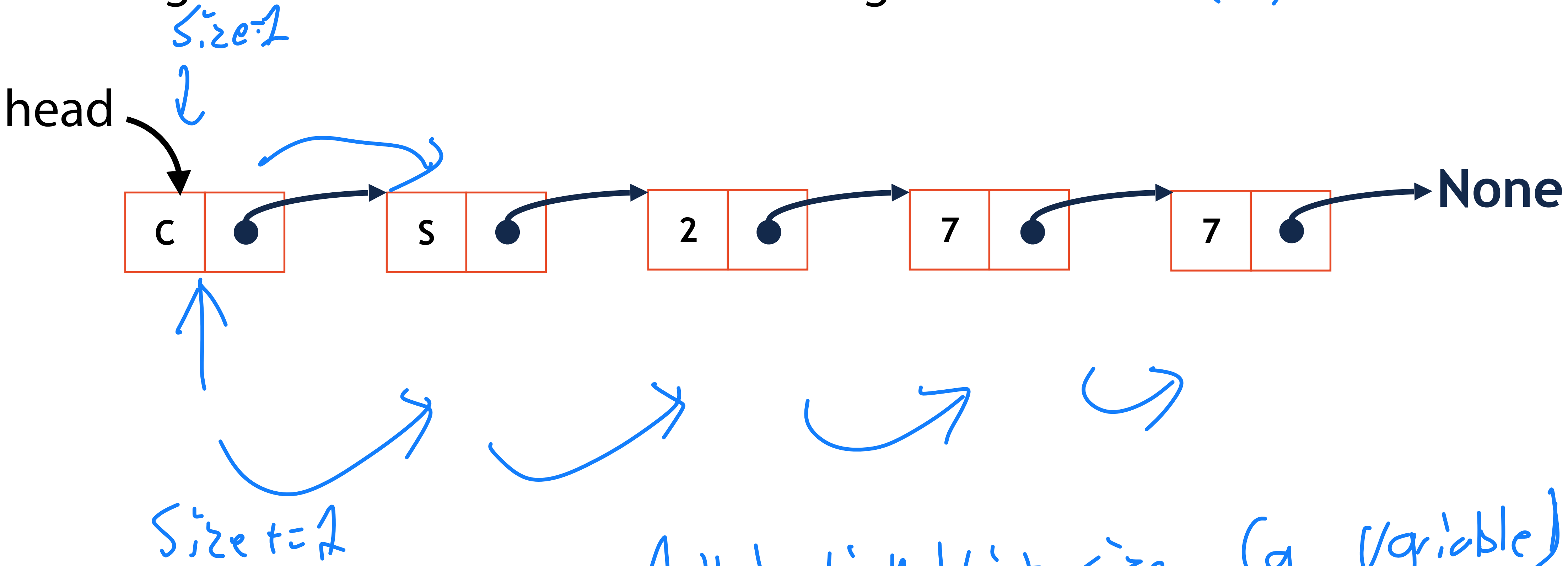
trivial way to become $O(1)$ → preprocess every input

↳ w/ infinite time & memory

store all possible answers

Thinking critically about lists: tradeoffs

Getting the size of a linked list has a Big O of: O(n)



Added linkedList.size (a variable)

↳ O(1) ← cost is storing size

Thinking critically about lists: tradeoffs

Specifically this case

Insert here is O(1)*

Does knowing our list is sorted change our Array Big O?

unsorted

2	7	5	9	7	14	1	0	8	X

find Min O(1)

X → O(n)

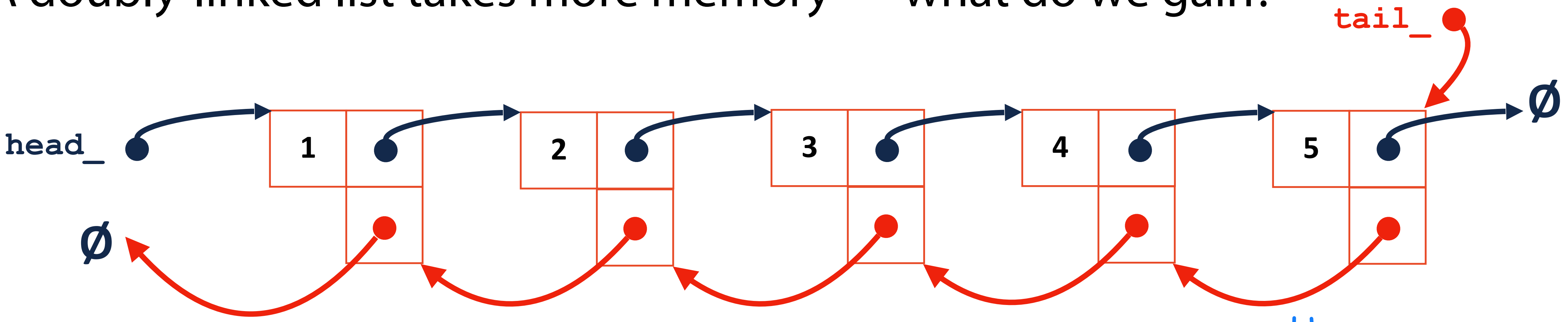
Find Max → O(1)

sorted

0	1	2	5	7	7	8	9	14	

Thinking critically about lists: tradeoffs

A doubly-linked list takes more memory — what do we gain?



walk backwards!

Increase in size

Node
↳ Data
↳ Next
↳ Prev

Thinking critically about lists: tradeoffs

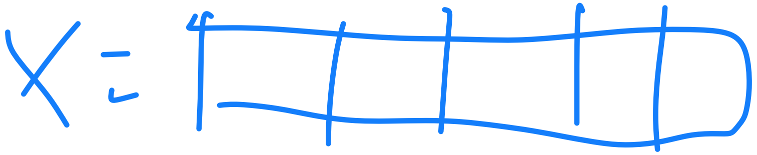
Consider carefully how data structures can be modified for a problem!

Lets see two examples of this:

I want a data structure that can add and remove in O(1).

I'm willing to 'trade away' a lot of utility to do this.

↳ I don't care about random access



X[2]

The Stack ADT

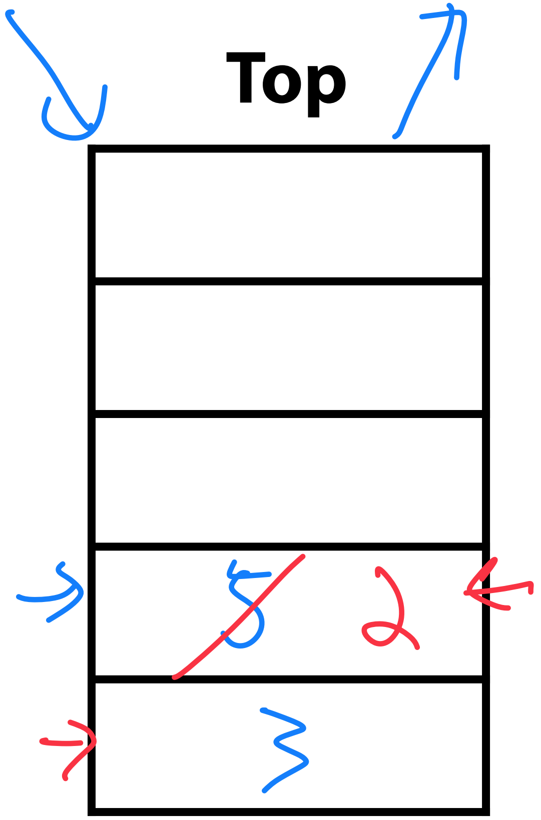
A **stack** stores an ordered collection of objects (like a list)

However you can only do three operations:

Push: Put an item on top of the stack

Pop: Remove the top item of the stack (and return it)

Top: Look at the value of the top item



```
push(3); push(5); top(); pop(); push(2)
```

↳ 5

Programming Toolbox: Stack

The stack is a **last in — first out** data structure (LIFO)

```
def reverse(inList):
    s = stack()
    for v in inList:
        s.push(v)

    out = []
    while not s.empty():
        out.append(s.pop())
    return out
```

Not a Python built-in!

`reverse([3, 4, 5, 6, 7, 8])`

stack s

8
7
6
5
4
3

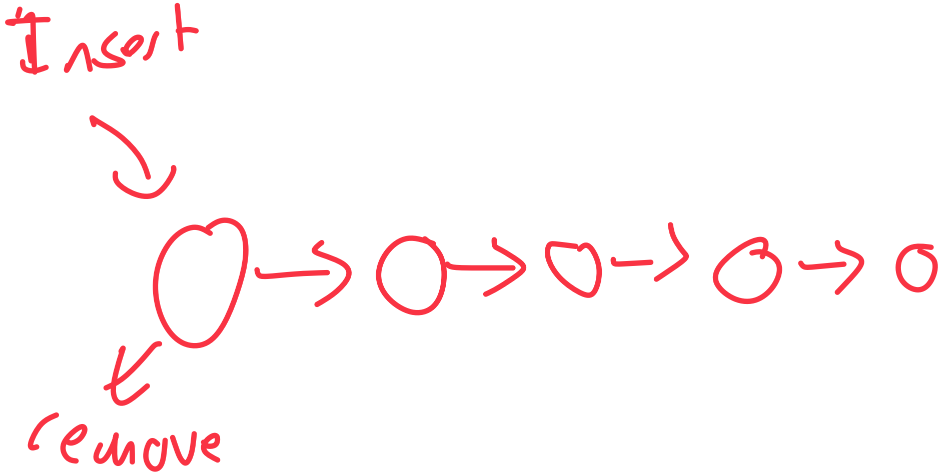
out	8	7	6	5	4	3

Stack Implementation (using Lists)

good at random access

	Singly Linked List	Array
Look up **arbitrary** location	O(n)	**O(1)**
Insert after **given** element	**O(1)**	O(n)
Remove after **given** element	**O(1)**	O(n)
Insert at **arbitrary** location	O(n)	O(n)
Remove at **arbitrary** location	O(n)	O(n)
Search for an input **value**	O(n)	O(n)

good at insert / remove in specific cases



Stack as a Linked List

`push(X)`

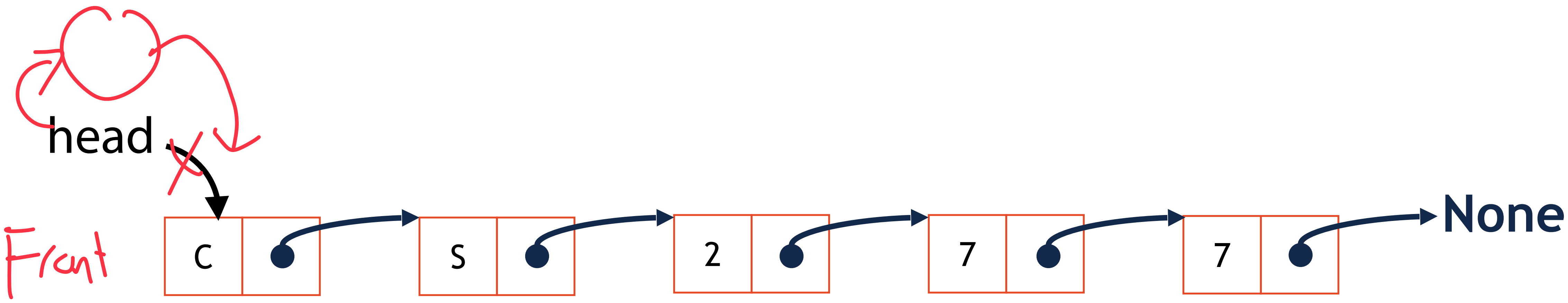
Push() adds the provided item to the top of my list

1) Make new Node(X, head)

next

O(1)

2) head = New Node



Stack as a Linked List

pop()

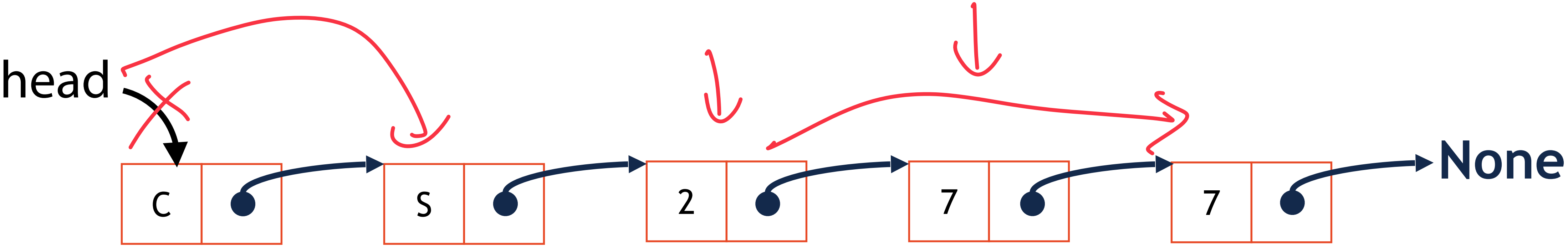
Pop() removes the top item from my list

1) head = head.next   O(1)

O(1) as well if we have curr

curr.next = curr.next.next

remove



Stack as a Linked List

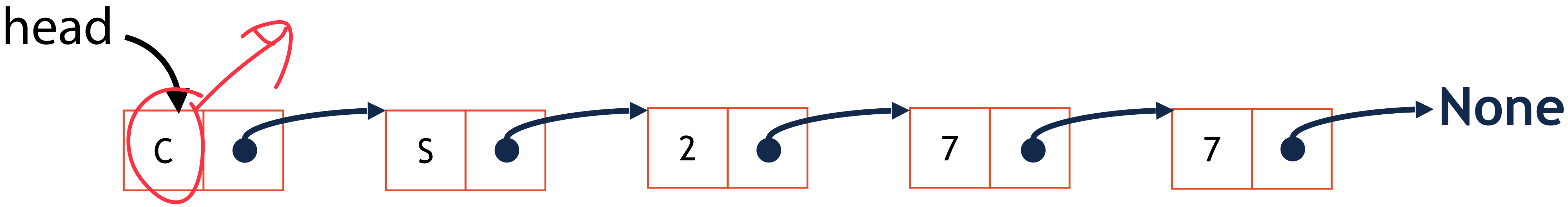
top()

Top() looks at the top item of the list

head.data

O(1)

.val



Stack Implementation

```
class Node:
    def __init__(self, value, next = None):
         self.val = value
         self.next = next

class stack:
    def __init__(self):
        self.head = None
        self.length = 0

    def push(self, value):
        self.length += 1
        newNode = Node(value)
        newNode.next = self.head
        self.head = newNode

    def __len__(self):
        return self.length

# class stack:
    def top(self):
        if self.length > 0:
            return self.head.val
        return None

    def pop(self):
        if (self.length > 0):
            self.length -= 1
            popped = self.head
            self.head = self.head.next
            return popped.val
        return None

# Some other support functions in code base
```

On your own: Stack Practice

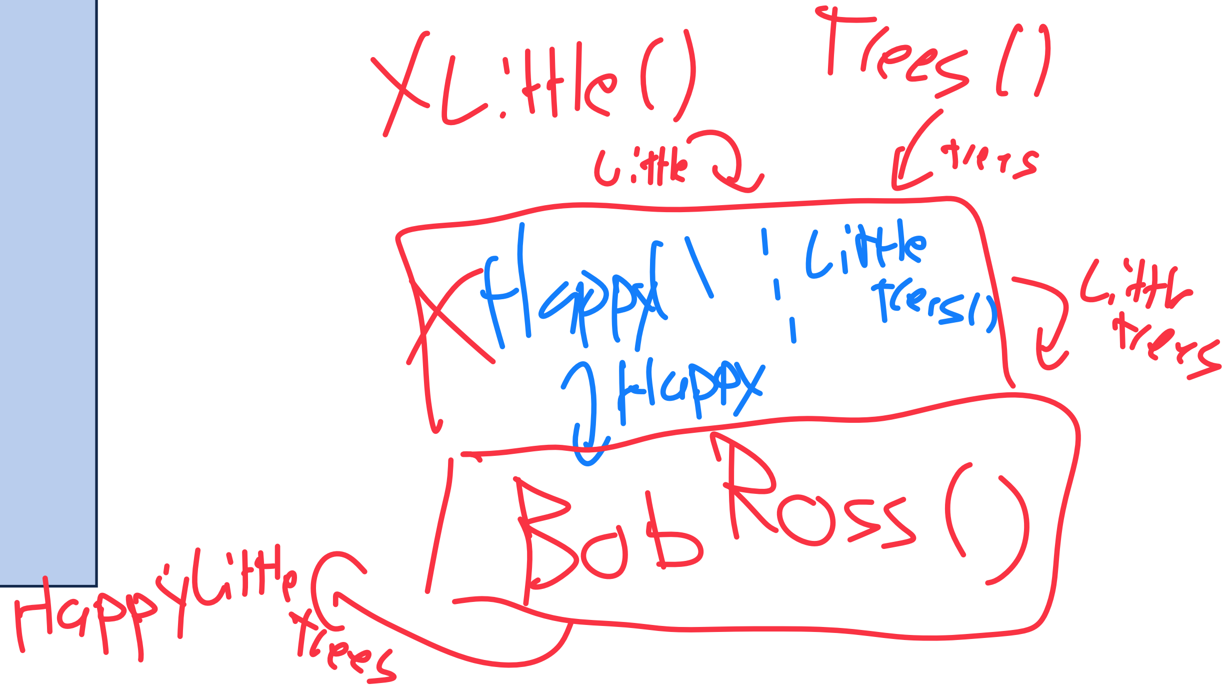
What will the stack look like as you run the following code?

Try it by hand and run the code to check!

```
s = stack()

print(s.empty())

for i in range(0, 10, 2):
    s.push(i)

print(s)

x = s.pop()
print(x, s)

print(len(s))

print(s.top())
s.pop()
print(s.top())

print(s.empty())
```

The Call Stack

```python
def Happy():
    print("Calling Happy!")
    return "Happy"

def Little():
    print("Calling Little!")
    return "Little"

def Trees():
    print("Calling Trees!")
    return "Trees"

def LittleTrees():
    print("Calling LittleTrees!")
    return Little() + Trees()

def BobRoss():
    print("Calling BobRoss!")
    return Happy() + LittleTrees()

print(BobRoss())


```



Queue Data Structure

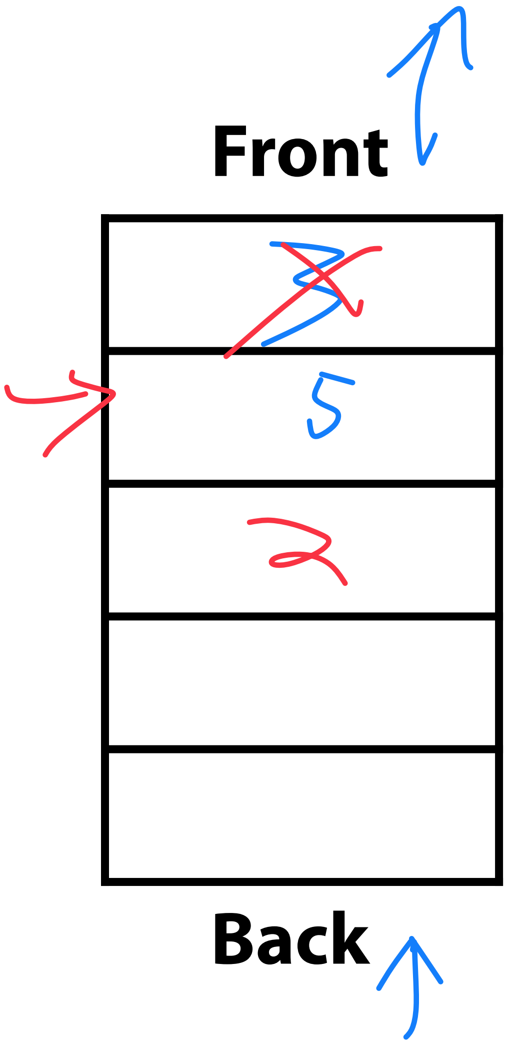
A **queue** stores an ordered collection of objects (like a list)

However you can only do ~~two~~ three operations:

Enqueue: Put an item at the back of the queue

Dequeue: Remove and return the front item of the queue

Front: Get value at front of queue



Front

Back

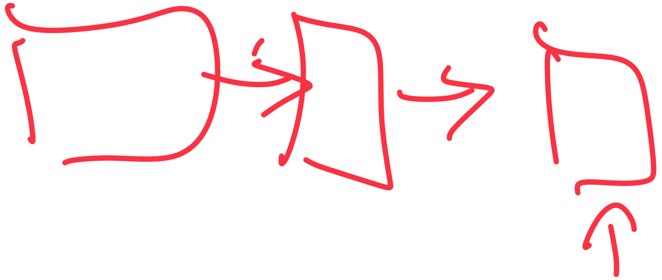
```
enqueue(3);  enqueue(5);  dequeue();  enqueue(2)
```

Queue Data Structure

The queue is a **first in — first out** data structure (FIFO)

What data structure excels at removing from the front?

Linked List



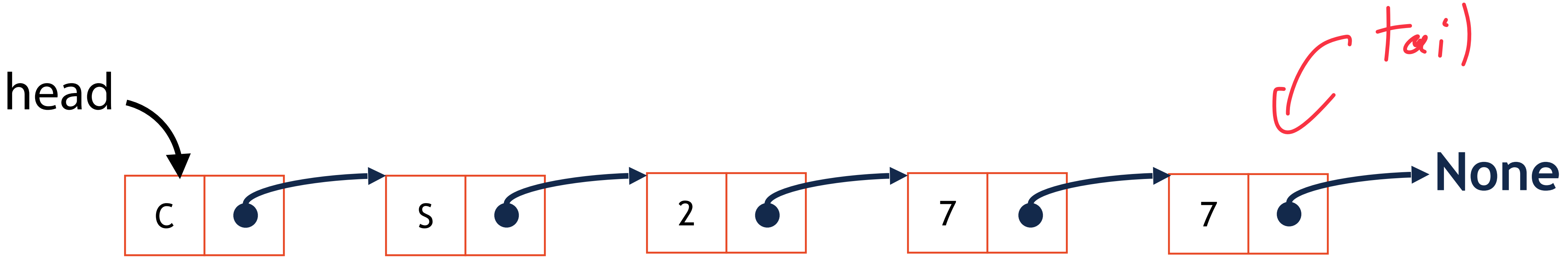
Can we make that same data structure good at inserting at the end? ??

↳ Can we make head variable for end of list?

Queue Data Structure

Once again, a linked list is a great implementation of a queue!

But we need one modification…



Linked List

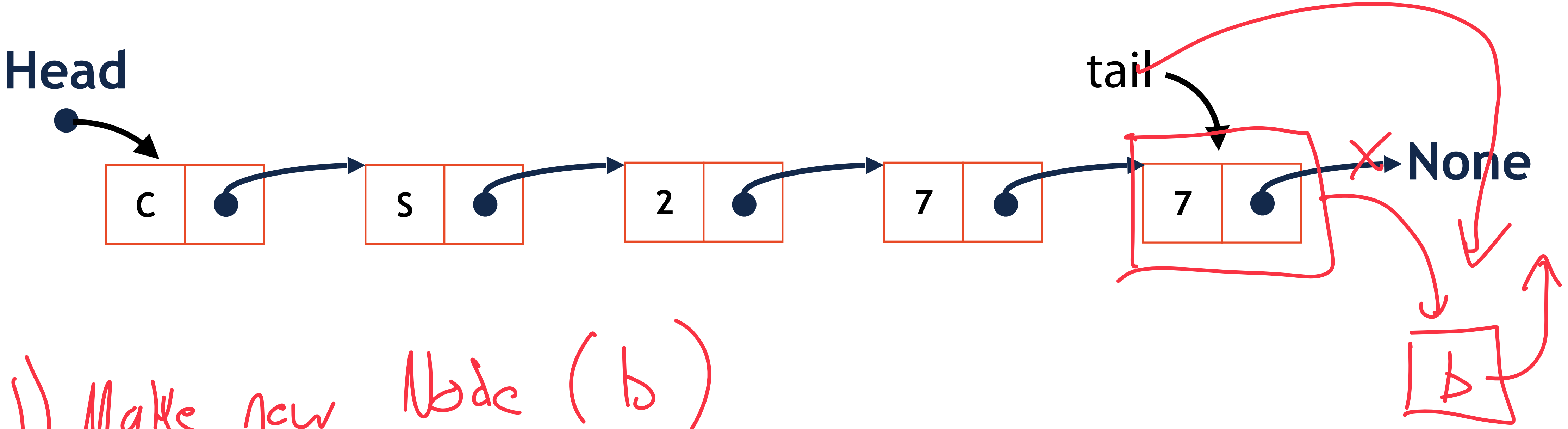
↳ head

↳ length

↳ tail - points to last item

Queue as Linked List

enqueue (b)



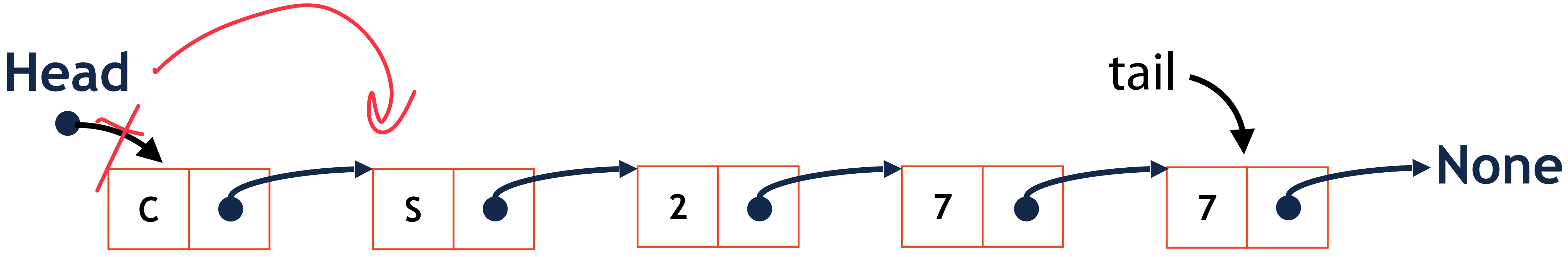
1) Make new Node (b)

2) tail.next = new Node

3) tail = new Node   # tail = tail.next

Queue as Linked List

`dequeue()`



↳ This is stack pop()!

Queue Implementation

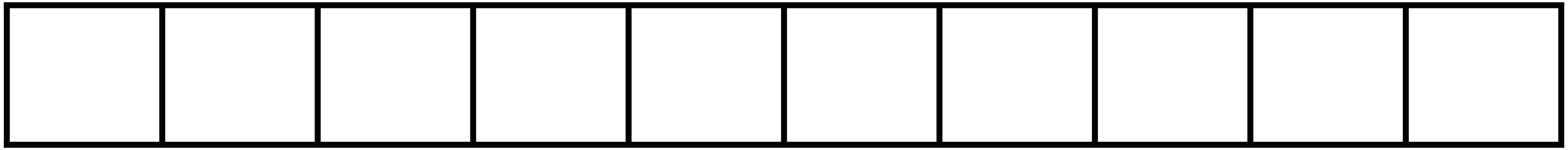
```
class Node:
    def __init__(self, value, next = None):
         self.val = value
         self.next = next

class queue:
    def __init__(self):
        self.length = 0
        self.head = None
        self.tail = None

    def enqueue(self, value):
        self.length += 1
        item = Node(value)
        if self.head == None:
            self.head = item
            self.tail = item
        else:
            self.tail.next = item
            self.tail = self.tail.next



# class queue:
    def dequeue(self):
        if self.length > 0:
            self.length -= 1
            item = self.head
            self.head = item.next

            if self.head == None:
                self.tail = None
            return item.val

        return None

    def front(self):
        if self.length > 0:
            return self.head.val
        return None

# Other support functions in code base
```

On your own: Queue Practice

```
q = queue()

print(q.empty())

for i in range(0,20, 2):
    q.enqueue(i)
print(q)

x = q.dequeue()
print(x, q)

print(len(q))

print(q.front())
q.dequeue()
print(q.front())

print(q.empty())
```

What will this code output?

Try it by hand and run the code to check!



The CS 277 Queue

Now you know how the CS 277 queue works!

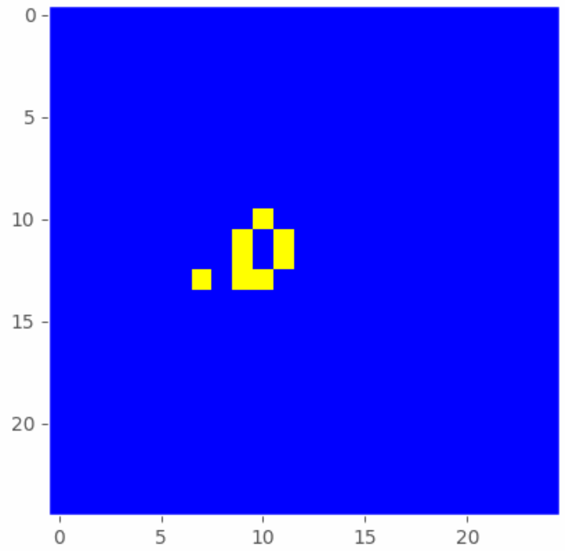
https://queue.illinois.edu/q/course/185

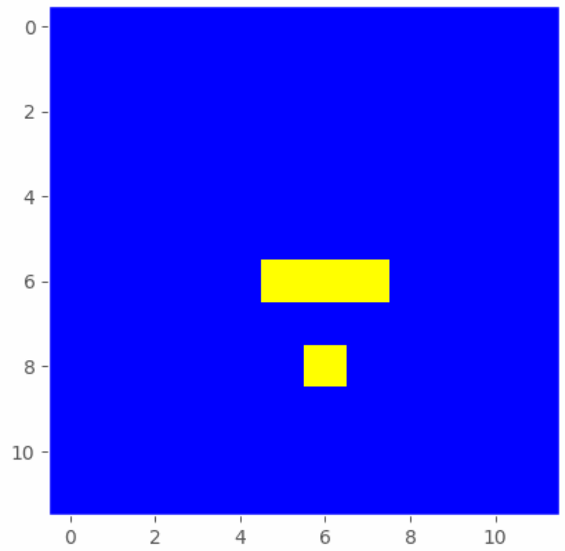
Cellular Automata — probably pushed back?

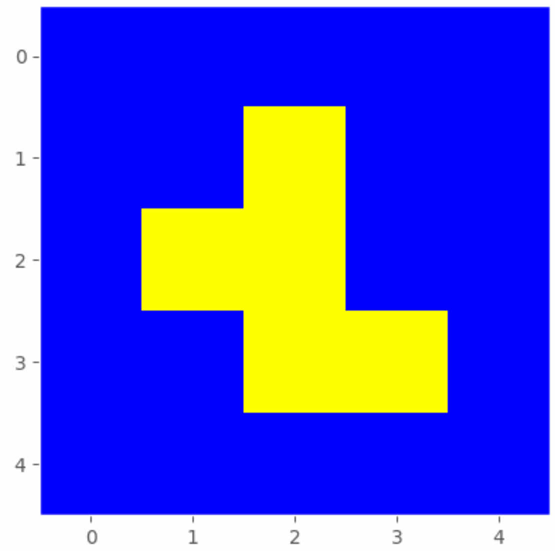
A computational model consisting of a **matrix** and a **set of rules**

Each iteration, the matrix changes based on its current state

There are a number of emergent behaviors that can be discovered!







Programming Practice: 2D Lists

Given a function that passes in three lists, make a 2D array storing all three.

See

How would I search a 2D list for a specific value?

code

Programming Practice: 2D Lists

What shape will this code produce?

5 x 5 list why?

```
outerList = []

for i in range(5):
    innerList = []

    for j in range(5):
        innerList.append(i+j)

    outerList.append(innerList)
```

# of rows

i

size of inner list

# of cols

i=0 [[i]

i=1 [x]

i=2 [x, x]

i=3 [. . . .]]

Programming Practice: 2D Lists

What values will this list produce?

	j=0	1	2	3	4
i=0	0	1	2	3	4
=1	1	2	3	4	5
2	2	3	4	5	6
3		-	. .	.	
4					

```
outerList = []

for i in range(5):
    innerList = []

    for j in range(5):
        innerList.append(i+j)

    outerList.append(innerList)
```

Programming Practice: 2D Lists

What are the indices of every value of 4 in this list? X[0][4]

X =

	0	1	2	3	4
0 →	0	1	2	3	4
	1	2	3	4	5
	2	3	4	5	6
	3	4	5	6	7
	4	5	6	7	8

1, 3

2, 2

3, 1

(0, 4)

row, col

```
outerList = []

for i in range(5):
    innerList = []

    for j in range(5):
        innerList.append(i+j)

    outerList.append(innerList)
```

Programming Practice: 2D Lists

What are the indices of the square around the point (x, y)?

0,0   x,y ≡ raw, col   col

0th value
top row [→ (0,0)

x-1, y-1	x-1, y	
y-1	(x, y) →	y+1 5
	7	8

row

↑ -1 row

↓ +1 row

-1 col ←   +1 col

Programming Tip: Make a quick example!

I want to make a fast matrix with the values:

1	2	3
4	5	6
7	8	9

```
count = 0
out = []
for i in range(3):
    tmp = []
    for j in range(3):
        tmp.append(count)
        count+=1
    out.append(tmp)

print(out[2][2])

print(out[2][1])

print(out[1][2])
```

```
import numpy as np

x = np.arange(9).reshape(3, 3)

print(x)
```

Programming Toolbox: NumPy

NumPy is optimized for multidimensional arrays of numbers

```
import numpy as np

# Convert list to np list
nl = np.array([1, 2, 3, 4, 5, 6])
print(nl)

# See list shape
print(nl.shape)

# Modify list shape
nl2 = nl.reshape(3, 2)

print(nl)
print(nl2)

# Create a new list
nl3 = np.arange(15).reshape(5, 3)
nl4 = np.zeros((2, 5))

print(nl3)
print(nl4)


```

np.array(<list>)

Convert a built-in list to a NumPy List

<nparray>.shape

Get the shape of the NumPy array

np.reshape(<row>, <col>)

If the list contains exactly row x col items

reshape the list to those dimensions

np.arange()

returns a NumPy array with range() values

np.zeros((<row>, col>))

Create a list of 0s of the provided shape

Programming Toolbox: NumPy

Basic operations are applied **elementwise** (to each item of a list)

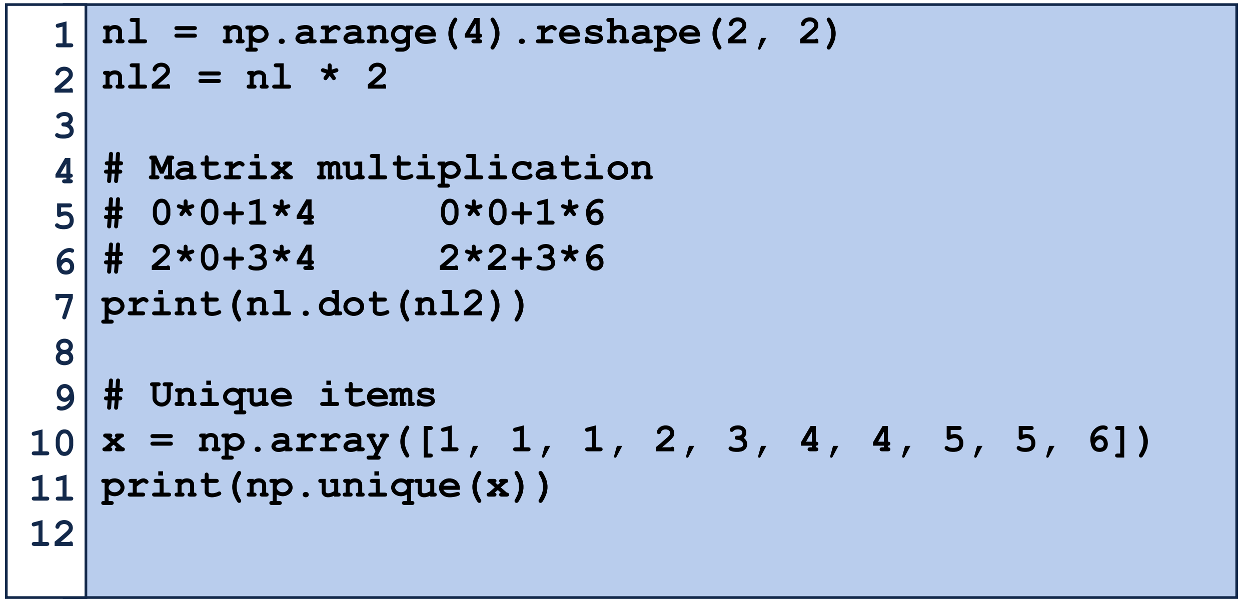
```
nl = np.arange(4).reshape(2, 2)

print(nl)

nl2 = nl * 2

print(nl2)

```

NumPy lists of equal size can also be added / subtracted / multiplied / etc…

```
m1 = np.arange(9).reshape(3,3)
print(m1)

m2 = np.arange(9, 0, -1).reshape(3,3)
print(m2)

print(m1+m2)

```

Programming Toolbox: NumPy

NumPy is huge and can do many things



```
nl = np.arange(4).reshape(2, 2)
nl2 = nl * 2

# Matrix multiplication
# 0*0+1*4      0*0+1*6
# 2*0+3*4      2*2+3*6
print(nl.dot(nl2))

# Unique items
x = np.array([1, 1, 1, 2, 3, 4, 4, 5, 5, 6])
print(np.unique(x))

```

Explore on your own: https://numpy.org/devdocs/

Next Week: Copying 2D Lists

What happens when we run the following code? Why?

```
orig = [ [1,2,3], [4, 5, 6]]

copy = orig[:]

orig[1][1]=9
copy[0][2]=7

print(orig)
print(copy)
```