Algorithms and Data Structures for Data Science

Lists

CS 277
Brad Solomon

September 9, 2021

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

lab_parsing Feedback

Average completion time looks to be ~3 hours

Not everyone found the lecture helpful

All students agreed lab matched learning objectives

Avg: 13.89 / 15

Planning out a problem

Write down the input and output (make sure you know it!)

Work out by hand the steps you need to get the output

Figure out how to do the necessary steps

Write the code — and then test it!

Reflecting on lab_parsing

Input:

```
bradsol,10,8,5,7,8
hdeep,10,8,5,7,8,9
```

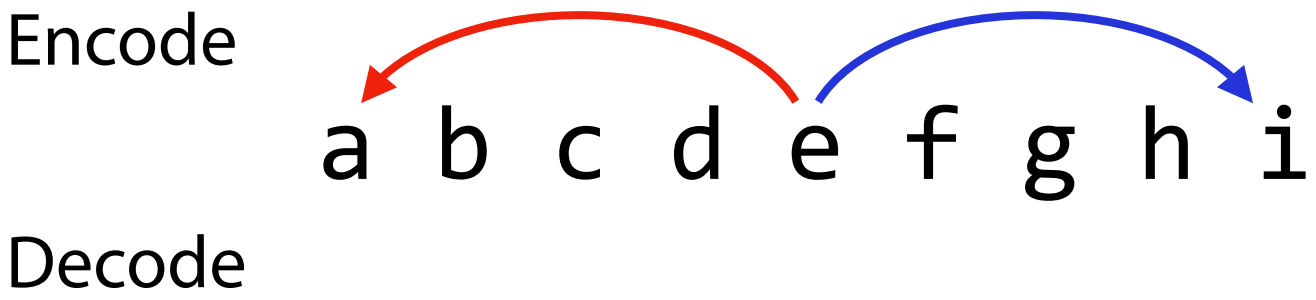
Output:

Open Dialogue about CS 277

lab_cipher adjustments

Vigenere Decode now extra credit

CaesarEncode / Decode significantly up-weighted



mp_racing adjustments

MPs designed as mini-data science projects

(Most) content is reasonable — clarity not so much

Future assignments will be simpler and shorter

PuzzleMovement extra credit — for now

mp_racing: loadMap()

Input:

```
0 0
1 1 1
1 0 1
1 1 1
```

_map.txt

Output:

mp_racing: loadCargo()

Input:

```
3
5
3
7
```

_cargo.txt

Output:

mp_racing: checkSquare()

Input:

Output:

mp_racing: checkAvail()

```
    def checkAvail(self):
        if self.curr == None:
            return []

        # racingBot's current position
        row = self.curr[0]
        col = self.curr[1]

        # The return list
        avail = []

        north = (0,0)

        if self.checkSquare(north):
            avail.append(north)


        return avail
```

Takeaway thoughts

There will be a proficiency exam in the near future

Course curriculum will slow down

Pacing and Content survey on Moodle

Learning Objectives

Define the list *abstract data type*

Discuss implementation strategies for lists

Debate which strategy is better (and how to judge ‘better’)

List Abstract Data Type

Values:

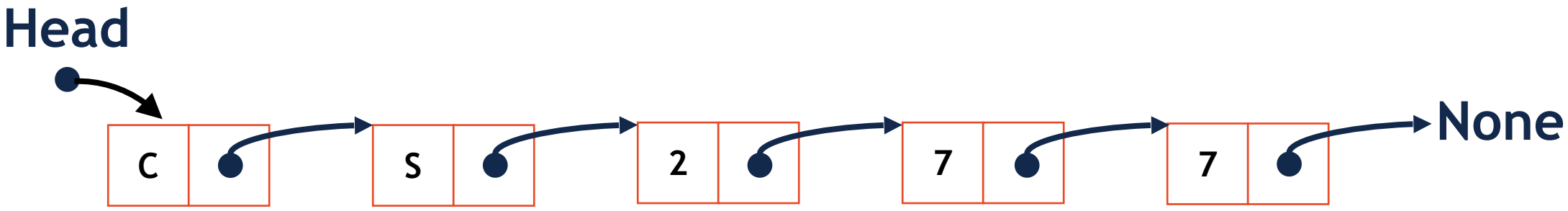
Functions:

List Implementations

1.

2.

Linked List

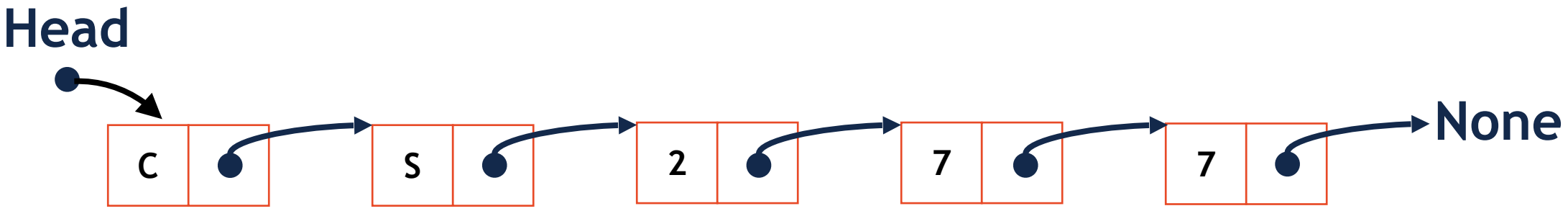


Creating a linked list

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

class linkedList:
    def __init__(self, head=None):
        self.head = head

if __name__ == '__main__':
    n1 = Node(3)
    n2 = Node(5)
    n3 = Node(7)

    n1.next = n2
    n2.next = n3
    n3.next = n1

    ll = linkedList(n1)
```

Linked List

Add (B)

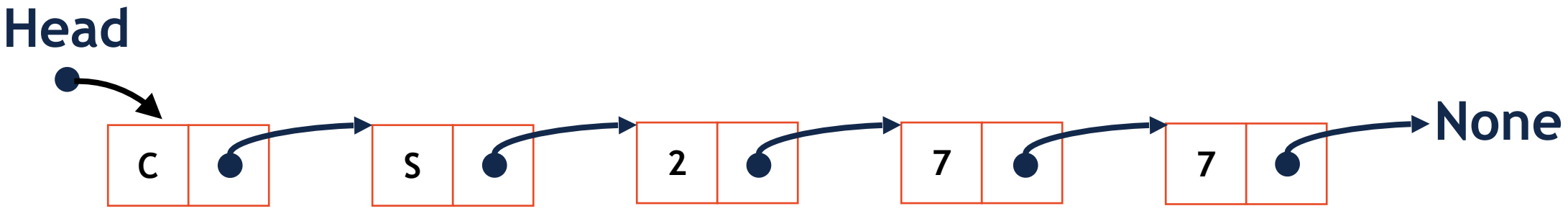


Linked List Add

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

class linkedList:
    def __init__(self, head=None):
        self.head = head

    def add(self, data):





if __name__ == '__main__':
    ll = linkedList()

    for i in range(3):
        ll.add(i)
```

Linked List

`Index(3)`

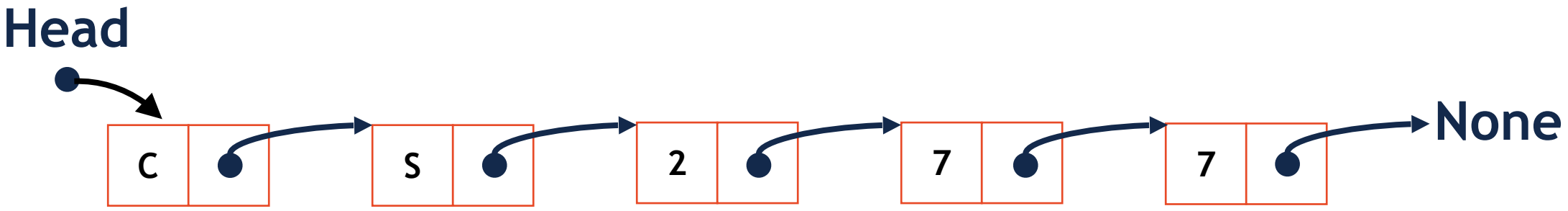


Linked List Index

```
class linkedList:

    def index(self, pos):
        curr = self.head
```

Linked List

Insert(X,2)

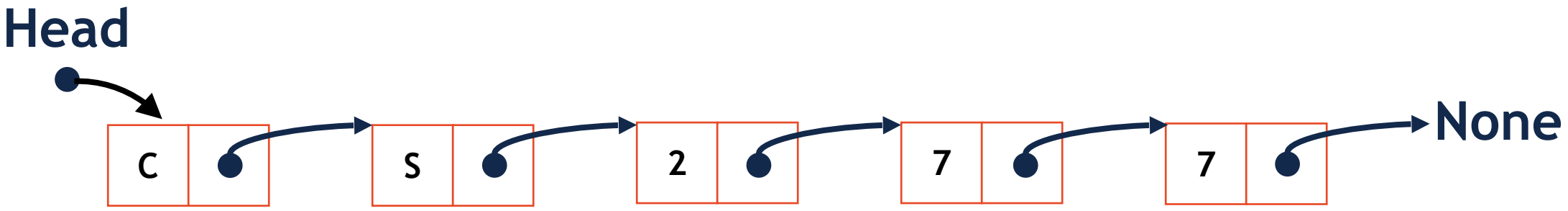


Linked List Insert

```
class linkedList:

    def insert(self, data, pos=0):
        if (pos == 0):
            self.add(data)
        else:
```

Linked List

Remove(7)

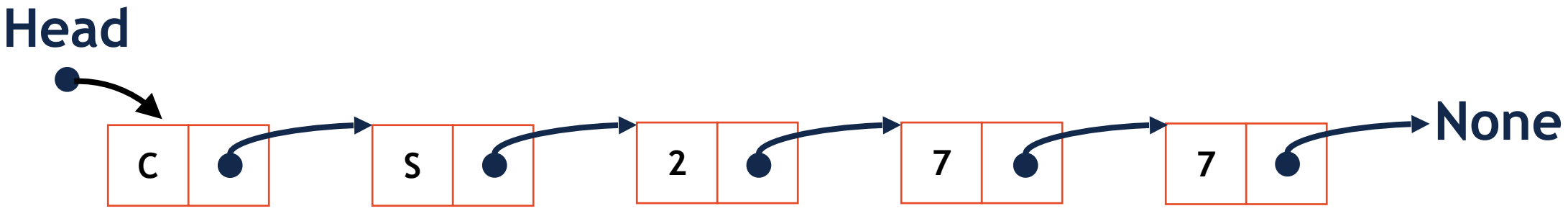


Linked List Remove

```
class linkedList:
    def remove(self, data):
        prev = None
        curr = self.head
```

Linked List

Pop(4)



Linked List Pop

```
class linkedList:
   def pop(self, pos):
```

Exercise 1

```
ll = linkedList()

for i in range(3):
    ll.add(i)


ll.insert(6,1)
ll.insert(3,1)
ll.insert(10,3)


ll.remove(6)
ll.remove(2)
ll.remove(12)


ll.pop(2)
ll.pop(0)
```

List Implementations

1. Linked List

2. Arrays

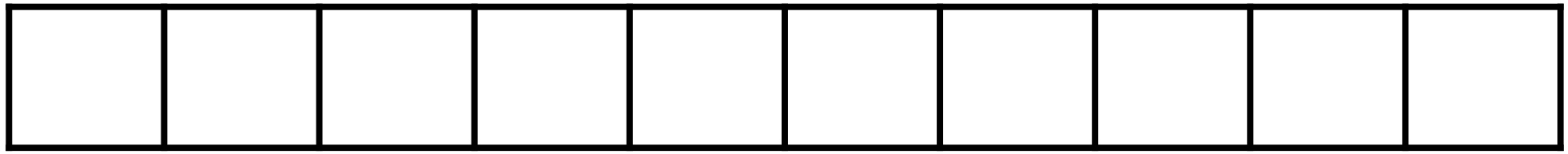
Array

Creating an array list

```
1 class arrayList:
2     def __init__(self, array=[]):
3         self.array = array
```

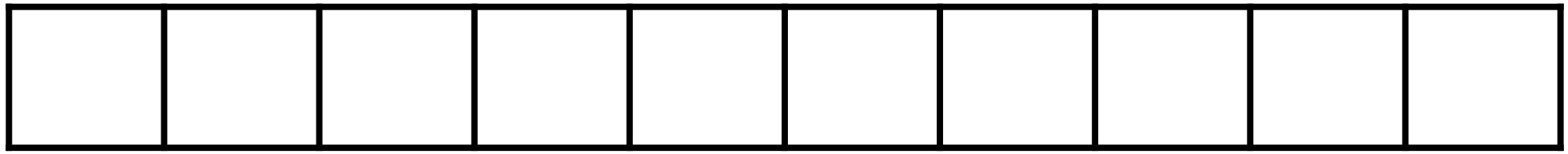
Array Add

```
1 def add(self, data):
2
3
```



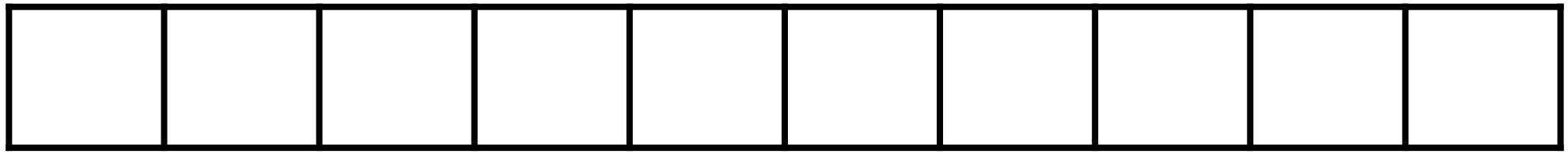
Array Index

```
1 def index(self, pos):
2
3
```



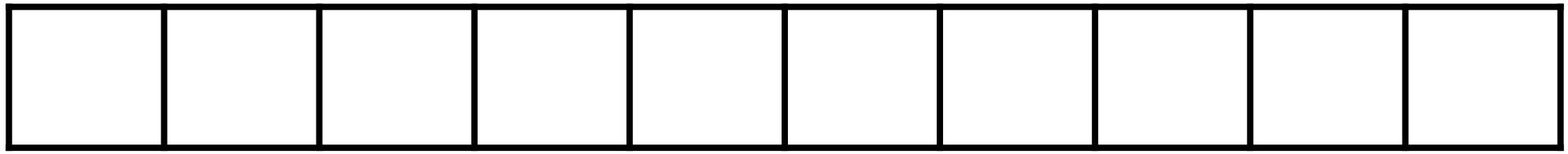
Array Insert

```
def insert(self, data, pos=0):


```



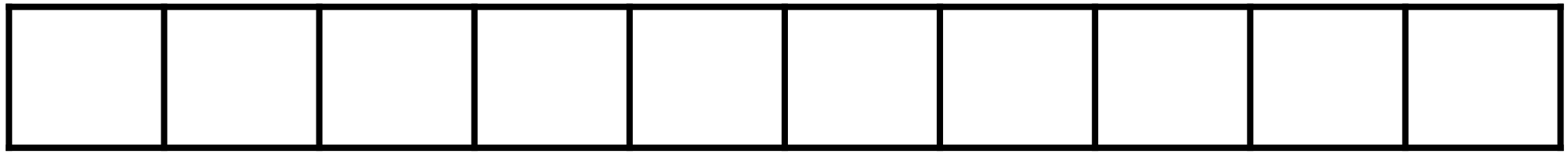
Array Remove

```
def remove(self, data):


```



Array Pop

```
def pop(self, pos):


```



Which implementation is better?

What do we care about when we write code?