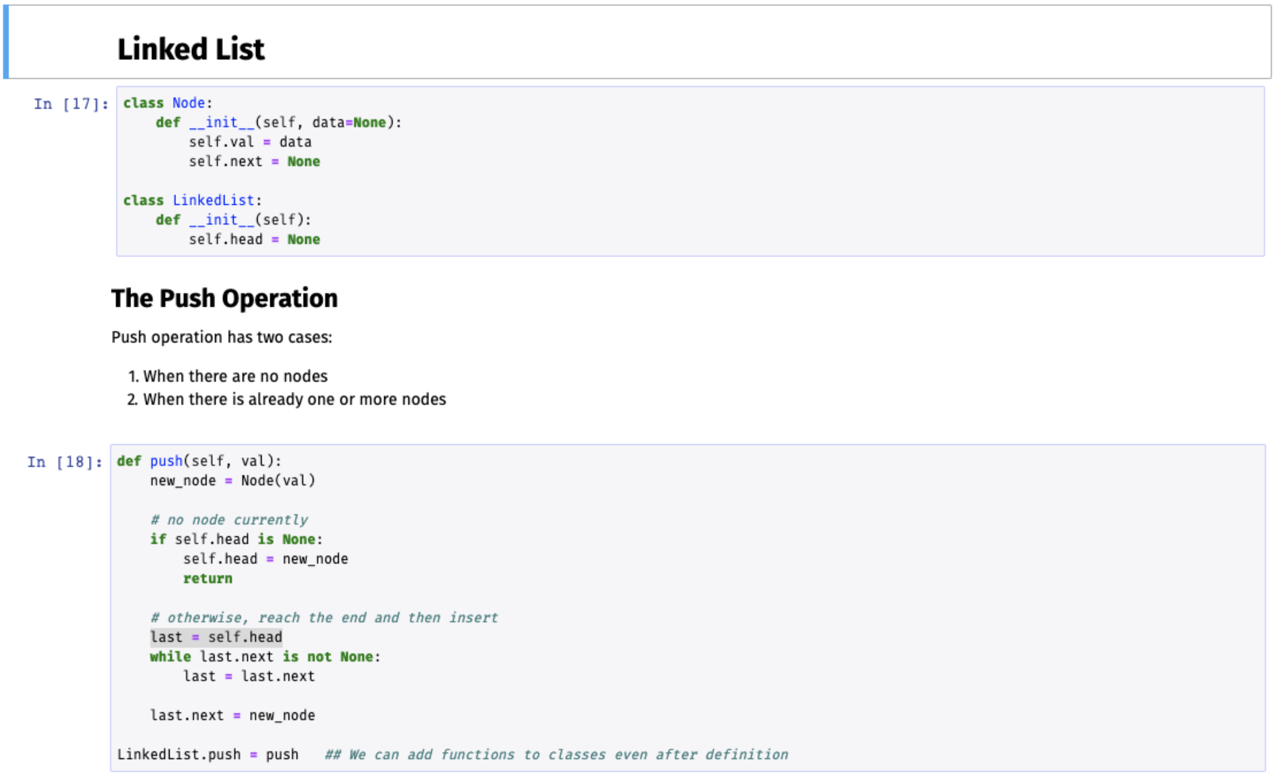
CS218 - Data Structures
FAST NUCES Peshawar Campus
Dr. Nauman (recluze.net)

August 26, 2019

1 Linked List in Python

Raster images of the notebook 04-linked-list



Linked List

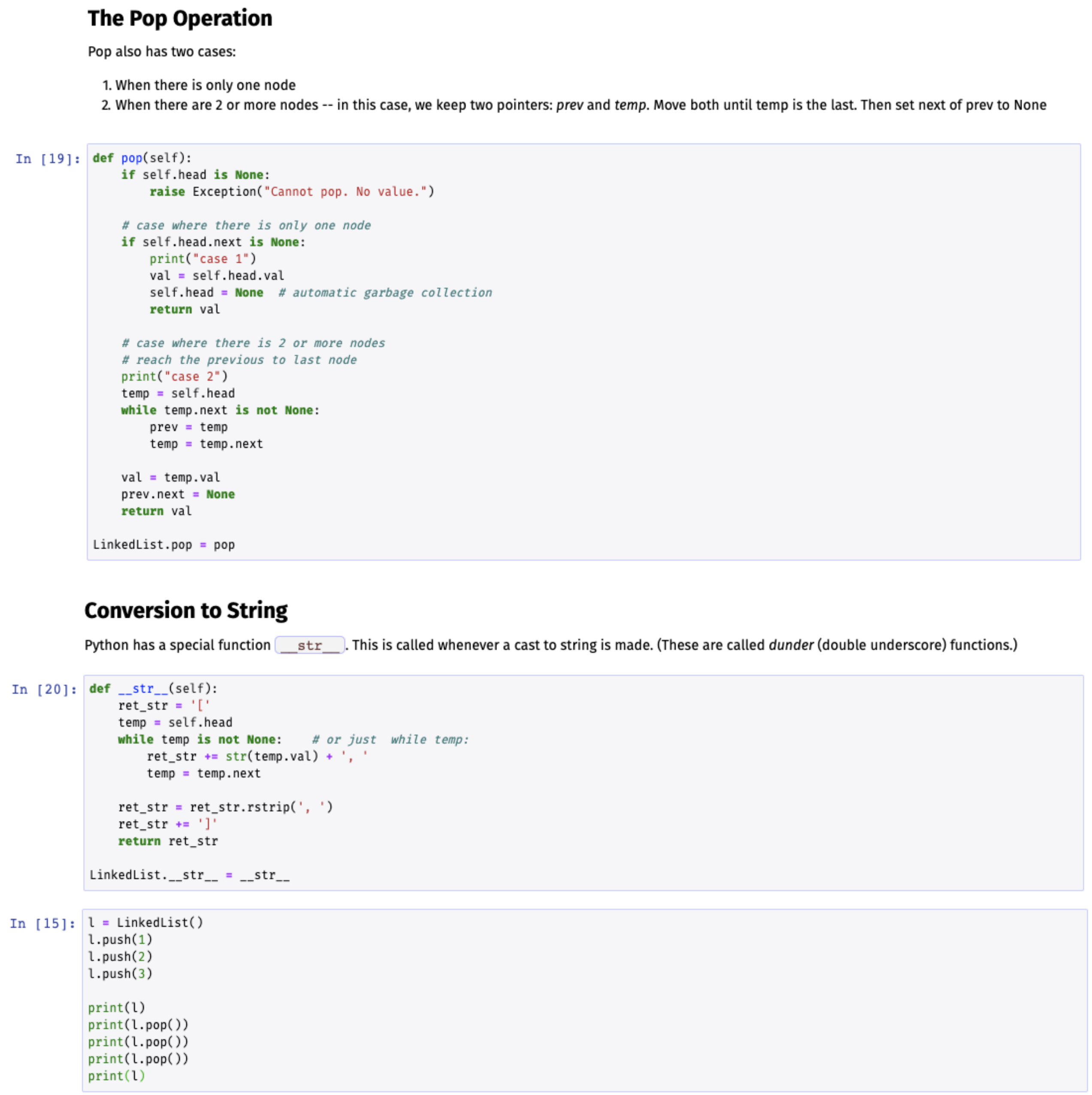
In [17]:
```
class Node:
    def __init__(self, data=None):
        self.val = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
```

The Push Operation

Push operation has two cases:

1. When there are no nodes
2. When there is already one or more nodes

In [18]:
```
def push(self, val):
    new_node = Node(val)

    # no node currently
    if self.head is None:
        self.head = new_node
        return

    # otherwise, reach the end and then insert
    last = self.head
    while last.next is not None:
        last = last.next

    last.next = new_node

LinkedList.push = push    ## We can add functions to classes even after definition
```



The Pop Operation

Pop also has two cases:

1. When there is only one node
2. When there are 2 or more nodes -- in this case, we keep two pointers: *prev* and *temp*. Move both until temp is the last. Then set next of prev to None

In [19]:

```python
def pop(self):
    if self.head is None:
        raise Exception("Cannot pop. No value.")

    # case where there is only one node
    if self.head.next is None:
        print("case 1")
        val = self.head.val
        self.head = None  # automatic garbage collection
        return val

    # case where there is 2 or more nodes
    # reach the previous to last node
    print("case 2")
    temp = self.head
    while temp.next is not None:
        prev = temp
        temp = temp.next

    val = temp.val
    prev.next = None
    return val

LinkedList.pop = pop
```

Conversion to String

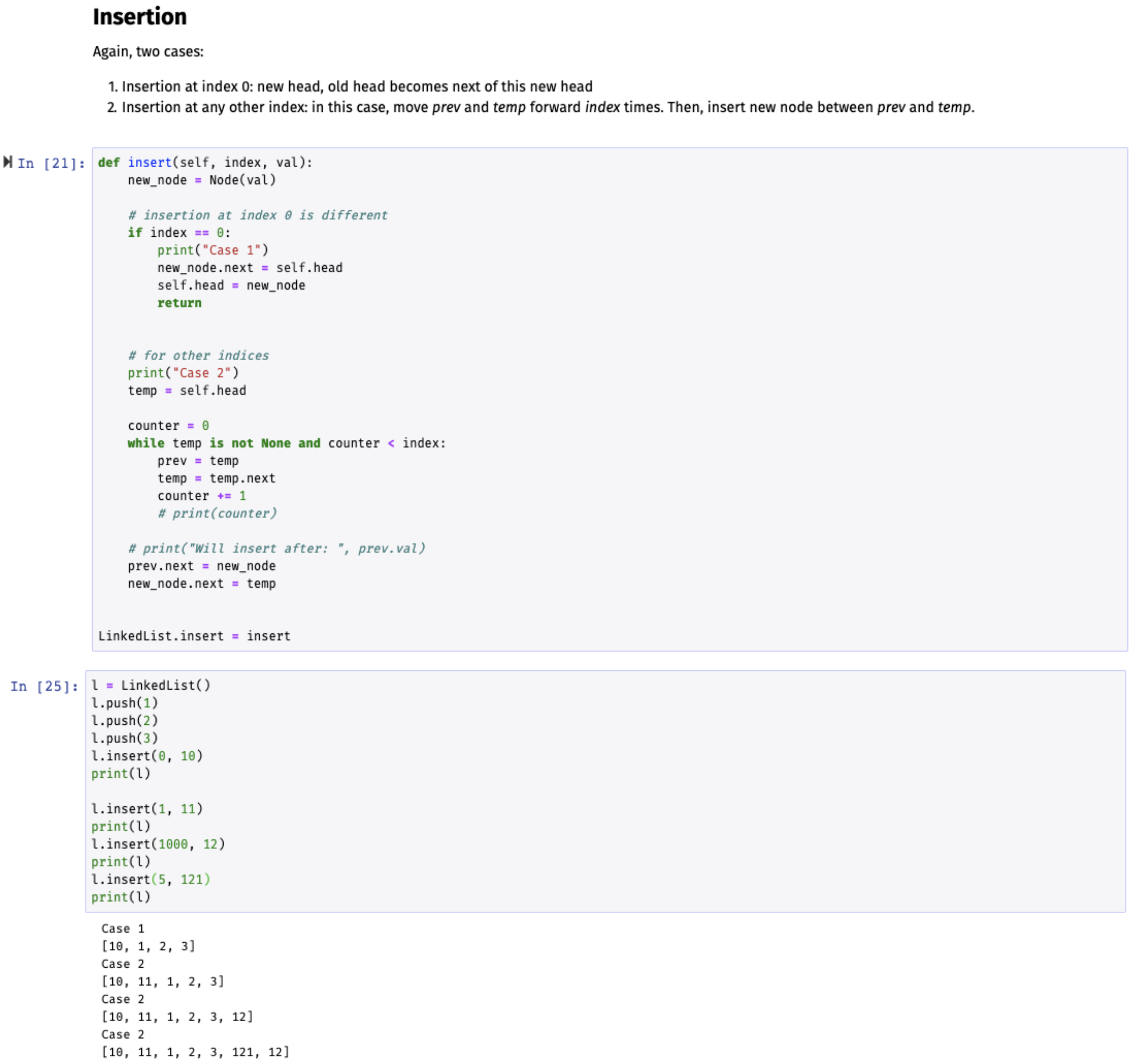
Python has a special function `__str__`. This is called whenever a cast to string is made. (These are called *dunder* (double underscore) functions.)

In [20]:

```python
def __str__(self):
    ret_str = '['
    temp = self.head
    while temp is not None:     # or just  while temp:
        ret_str += str(temp.val) + ', '
        temp = temp.next

    ret_str = ret_str.rstrip(', ')
    ret_str += ']'
    return ret_str

LinkedList.__str__ = __str__
```

In [15]:

```python
l = LinkedList()
l.push(1)
l.push(2)
l.push(3)

print(l)
print(l.pop())
print(l.pop())
print(l.pop())
print(l)
```



Insertion

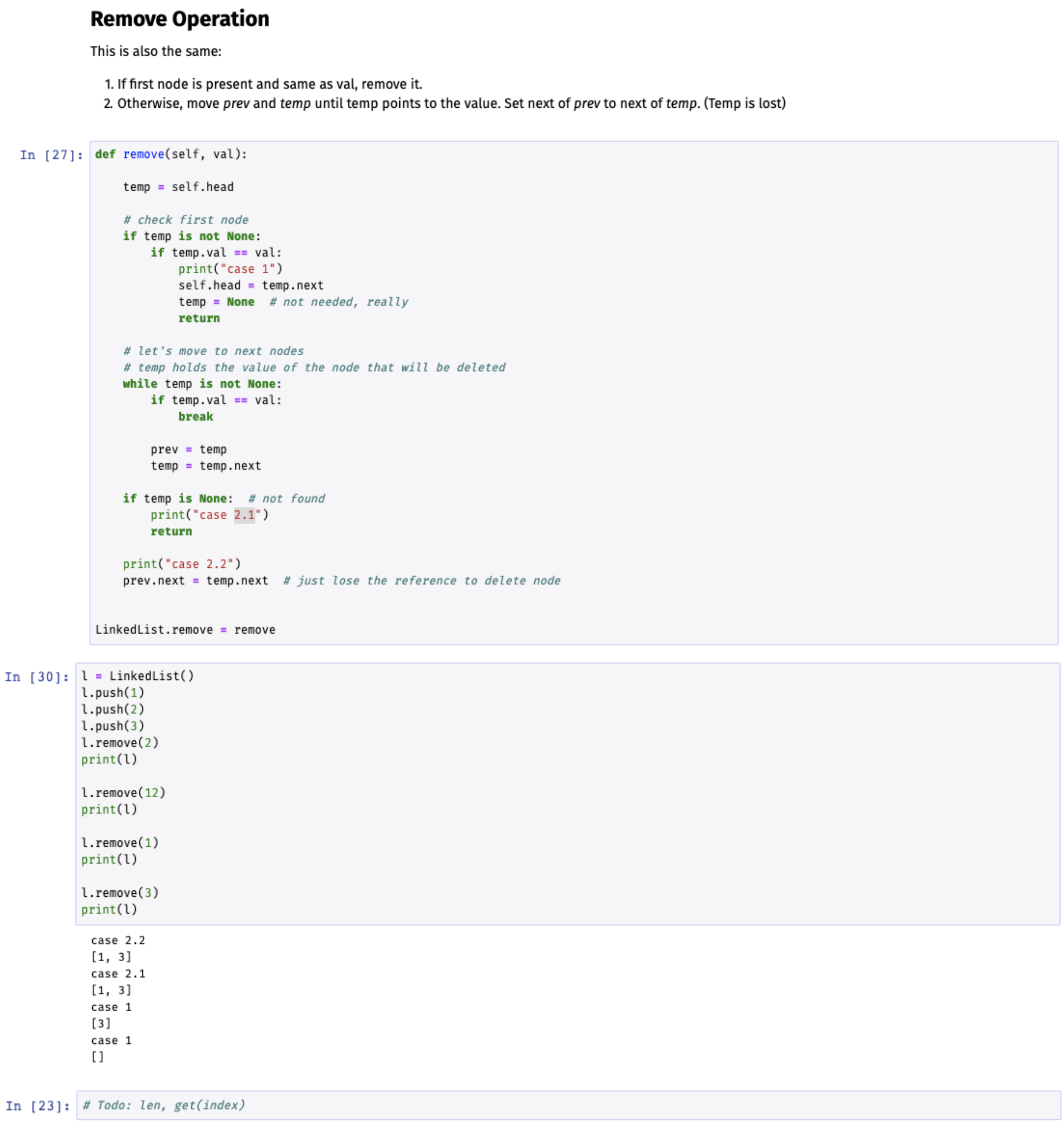
Again, two cases:

1. Insertion at index 0: new head, old head becomes next of this new head
2. Insertion at any other index: in this case, move *prev* and *temp* forward *index* times. Then, insert new node between *prev* and *temp*.

In [21]:
```
def insert(self, index, val):
    new_node = Node(val)

    # insertion at index 0 is different
    if index == 0:
        print("Case 1")
        new_node.next = self.head
        self.head = new_node
        return


    # for other indices
    print("Case 2")
    temp = self.head

    counter = 0
    while temp is not None and counter < index:
        prev = temp
        temp = temp.next
        counter += 1
        # print(counter)

    # print("Will insert after: ", prev.val)
    prev.next = new_node
    new_node.next = temp


LinkedList.insert = insert
```

In [25]:
```
l = LinkedList()
l.push(1)
l.push(2)
l.push(3)
l.insert(0, 10)
print(l)

l.insert(1, 11)
print(l)
l.insert(1000, 12)
print(l)
l.insert(5, 121)
print(l)
```

```
Case 1
[10, 1, 2, 3]
Case 2
[10, 11, 1, 2, 3]
Case 2
[10, 11, 1, 2, 3, 12]
Case 2
[10, 11, 1, 2, 3, 121, 12]
```



Remove Operation

This is also the same:

1. If first node is present and same as val, remove it.
2. Otherwise, move *prev* and *temp* until temp points to the value. Set next of *prev* to next of *temp*. (Temp is lost)

In [27]:
```
def remove(self, val):

    temp = self.head

    # check first node
    if temp is not None:
        if temp.val == val:
            print("case 1")
            self.head = temp.next
            temp = None  # not needed, really
            return

    # let's move to next nodes
    # temp holds the value of the node that will be deleted
    while temp is not None:
        if temp.val == val:
            break

        prev = temp
        temp = temp.next

    if temp is None:  # not found
        print("case 2.1")
        return

    print("case 2.2")
    prev.next = temp.next  # just lose the reference to delete node


LinkedList.remove = remove
```

In [30]:
```
l = LinkedList()
l.push(1)
l.push(2)
l.push(3)
l.remove(2)
print(l)

l.remove(12)
print(l)

l.remove(1)
print(l)

l.remove(3)
print(l)
```

```
case 2.2
[1, 3]
case 2.1
[1, 3]
case 1
[3]
case 1
[]
```

In [23]:
```
# Todo: len, get(index)
```