

Lecture 16, 09 October 2025

Linked lists

- Our current definition, with iterative `append()`

```
In [1]: class List:
    def __init__(self, initlist = []):
        self.value = None
        self.next = None
        for x in initlist:
            self.append(x)
        return

    def isempty(self):
        return(self.value == None)

    def append(self, v):
        if self.isempty():
            self.value = v
            return

        temp = self
        while temp.next != None:
            temp = temp.next

        temp.next = List()
        temp.next.value = v
        return

    def insert(self, v):
        if self.isempty():
            self.value = v
            return

        newnode = List()
        newnode.value = v

        # Exchange values in self and newnode
        (self.value, newnode.value) = (newnode.value, self.value)

        # Switch links
        (self.next, newnode.next) = (newnode, self.next)

        return

    def __str__(self):
        # Iteratively create a Python list from linked list
        # and convert that to a string
        selflist = []
        if self.isempty():
            return(str(selflist))

        temp = self
```

```

selflist.append(temp.value)

while temp.next != None:
    temp = temp.next
    selflist.append(temp.value)

return(str(selflist))

```

- Add recursive `append()`
 - `appendi()`, iterative
 - `appendr()`, recursive
 - Dummy `append()` that calls either `appendi()` or `appendr()`
 - To avoid problems with `__init__`, `__str__`

```

In [2]: class List:
def __init__(self, initlist = []):
    self.value = None
    self.next = None
    for x in initlist:
        self.append(x)
    return

def isempty(self):
    return(self.value == None)

def appendi(self, v): # append, iterative
    if self.isempty():
        self.value = v
        return

    temp = self
    while temp.next != None:
        temp = temp.next

    temp.next = List()
    temp.next.value = v
    return

def appendr(self, v): # append, recursive
    if self.isempty():
        self.value = v
    elif self.next == None:
        self.next = List()
        self.next.value = v
    else:
        self.next.appendr(v)
    return

def append(self, v): # Could point to appendi or appendr
    self.appendr(v)
    return

def insert(self, v):
    if self.isempty():
        self.value = v
        return

```

```

newnode = List()
newnode.value = v

# Exchange values in self and newnode
(self.value, newnode.value) = (newnode.value, self.value)

# Switch links
(self.next, newnode.next) = (newnode, self.next)

return

def __str__(self):
    # Iteratively create a Python list from linked list
    # and convert that to a string
    selflist = []
    if self.isempty():
        return(str(selflist))

    temp = self
    selflist.append(temp.value)

    while temp.next != None:
        temp = temp.next
        selflist.append(temp.value)

    return(str(selflist))

```

```

In [3]: l = List()
        for i in range(20,0,-1):
            l.appendr(i)
        print(l)

```

```
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

- Some performance measurements

```

In [4]: import time

```

- Iterative append, quadratic complexity

```

In [5]: for i in range(1,5):
        unit = 1000
        l1 = List()
        start = time.perf_counter()
        for j in range(i*unit):
            l1.appendi(j)
        elapsed = time.perf_counter() - start
        print(i*unit,elapsed)

```

```

1000 0.037408503994811326
2000 0.13884098699782044
3000 0.25658054200175684
4000 0.41125536999606993

```

- Recursive append, also quadratic, but 5x overhead due to recursive calls

```
In [6]: for i in range(1,5):
        unit = 1000
        l1 = List()
        start = time.perf_counter()
        for j in range(i*unit):
            l1.appendr(j)
        elapsed = time.perf_counter() - start
        print(i*unit,elapsed)
```

```
1000 0.15295926301041618
2000 0.5776570989983156
```

```
-----
-
RecursionError                                Traceback (most recent call las
t)
Cell In[6], line 6
      4 start = time.perf_counter()
      5 for j in range(i*unit):
----> 6     l1.appendr(j)
      7 elapsed = time.perf_counter() - start
      8 print(i*unit,elapsed)

Cell In[2], line 32, in List.appendr(self, v)
     30     self.next.value = v
     31 else:
----> 32     self.next.appendr(v)
     33 return

Cell In[2], line 32, in List.appendr(self, v)
     30     self.next.value = v
     31 else:
----> 32     self.next.appendr(v)
     33 return

[... skipping similar frames: List.appendr at line 32 (2973 times)]

Cell In[2], line 32, in List.appendr(self, v)
     30     self.next.value = v
     31 else:
----> 32     self.next.appendr(v)
     33 return

Cell In[2], line 26, in List.appendr(self, v)
     25 def appendr(self,v): # append, recursive
----> 26     if self.isempty():
     27         self.value = v
     28     elif self.next == None:

RecursionError: maximum recursion depth exceeded
```

- Enhance recursion limit
 - $2^{31} - 1$ is maximum allowed

```
In [7]: import sys
        sys.setrecursionlimit(2**31-1)
```

```
In [8]: for i in range(1,5):
        unit = 1000
```

```

l1 = List()
start = time.perf_counter()
for j in range(i*unit):
    l1.append(j)
elapsed = time.perf_counter() - start
print(i*unit,elapsed)

```

```

1000 0.1520825240004342
2000 0.59965024900157
3000 1.326439897005912
4000 2.4543402420094935

```

- Add `delete()`

```

In [9]: class List:
    def __init__(self,initlist = []):
        self.value = None
        self.next = None
        for x in initlist:
            self.append(x)
        return

    def isempty(self):
        return(self.value == None)

    def appendi(self,v):    # append, iterative
        if self.isempty():
            self.value = v
            return

        temp = self
        while temp.next != None:
            temp = temp.next

        temp.next = List()
        temp.next.value = v
        return

    def appendr(self,v):    # append, recursive
        if self.isempty():
            self.value = v
        elif self.next == None:
            self.next = List([v])
        else:
            self.next.appendr(v)
        return

    def append(self,v):
        self.appendr(v)
        return

    def insert(self,v):
        if self.isempty():
            self.value = v
            return

        newnode = List()
        newnode.value = v

```

```

        # Exchange values in self and newnode
        (self.value, newnode.value) = (newnode.value, self.value)

        # Switch links
        (self.next, newnode.next) = (newnode, self.next)

    return

def delete(self,v):    # delete, recursive
    if self.isempty():
        return

    if self.value == v:
        if self.next != None:
            self.value = self.next.value
            self.next = self.next.next
        else:
            self.value = None
        return
    else:
        if self.next != None:
            self.next.delete(v)
            # Ensure that there is no empty node at the end of the li
            if self.next.value == None:
                self.next = None

    return

def __str__(self):
    # Iteratively create a Python list from linked list
    # and convert that to a string
    selflist = []
    if self.isempty():
        return(str(selflist))

    temp = self
    selflist.append(temp.value)

    while temp.next != None:
        temp = temp.next
        selflist.append(temp.value)

    return(str(selflist))

```

```

In [10]: l = List(list(range(100)))
         print(l)

```

```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 3
9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 7
6, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
95, 96, 97, 98, 99]

```

```

In [11]: l.delete(1)
         print(l)

```

```
[0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

```
In [12]: l.delete(0)
         print(l)
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

```
In [13]: l.delete(99)
         print(l)
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98]
```

Exception handling

- Example with `input()`
- Python statement `pass` is a do-nothing placeholder. Cannot omit `except:` block, nor can you have an empty block.

```
In [14]: invalid = True
         while (invalid):
             try:
                 xstr = input("Enter a number: ")
                 xint = int(xstr)
                 invalid = False
             except:
                 pass
         print(xint)
```

-99

```
In [15]: invalid = True
         tryagain = False
         while (invalid):
             try:
                 if tryagain:
                     print("Try again:")
                 xstr = input("Enter a number: ")
                 xint = int(xstr)
                 invalid = False
             except:
                 tryagain = True
         print(xint)
```

Try again:
Try again:
-919

- Catch a specific type of exception

```
In [16]: int("abc")
```

```
-----  
-  
ValueError                                Traceback (most recent call las  
t)  
Cell In[16], line 1  
----> 1 int( )  
  
ValueError: invalid literal for int() with base 10: 'abc'
```

```
In [17]: invalid = True  
while (invalid):  
    try:  
        xstr = input("Enter a number: ")  
        xint = int(xstr)  
        invalid = False  
    except ValueError:  
        pass  
print(xint)
```

748

- Raising an exception in `List()`
- Inserting a negative value raises `ValueError`
- Add negative value to error message

```
In [18]: class List:  
    def __init__(self, initlist = []):  
        self.value = None  
        self.next = None  
        for x in initlist:  
            self.append(x)  
        return  
  
    def isempty(self):  
        return(self.value == None)  
  
    def appendi(self, v):    # append, iterative  
        if v < 0:  
            raise ValueError("Negative input: " + str(v))  
        if self.isempty():  
            self.value = v  
            return  
  
        temp = self  
        while temp.next != None:  
            temp = temp.next  
  
        temp.next = List()  
        temp.next.value = v  
        return
```

```

def appendr(self,v):    # append, recursive
    if v < 0:
        raise ValueError("Negative input: " + str(v))
    if self.isempty():
        self.value = v
    elif self.next == None:
        self.next = List([v])
    else:
        self.next.appendr(v)
    return

def append(self,v):
    self.appendr(v)
    return

def insert(self,v):
    if v < 0:
        raise ValueError("Negative input: " + str(v))
    if self.isempty():
        self.value = v
        return

    newnode = List()
    newnode.value = v

    # Exchange values in self and newnode
    (self.value, newnode.value) = (newnode.value, self.value)

    # Switch links
    (self.next, newnode.next) = (newnode, self.next)

    return

def delete(self,v):    # delete, recursive
    if self.isempty():
        return

    if self.value == v:
        self.value = None
        if self.next != None:
            self.value = self.next.value
            self.next = self.next.next
        return
    else:
        if self.next != None:
            self.next.delete(v)
            if self.next.value == None:
                self.next = None
        return

def __str__(self):
    # Iteratively create a Python list from linked list
    # and convert that to a string
    selflist = []
    if self.isempty():
        return(str(selflist))

    temp = self
    selflist.append(temp.value)

```

```

while temp.next != None:
    temp = temp.next
    selflist.append(temp.value)

return(str(selflist))

```

```

In [19]: l = List([1,-2,3])
         print(l)

```

```

-----
-
ValueError                                Traceback (most recent call last)
Cell In[19], line 1
----> 1 l = List([1,-2,3])
      2 print(l)

Cell In[18], line 6, in List.__init__(self, initlist)
      4 self.next = None
      5 for x in initlist:
----> 6     self.append(x)
      7 return

Cell In[18], line 39, in List.append(self, v)
      38 def append(self,v):
----> 39     self.appendr(v)
      40     return

Cell In[18], line 29, in List.appendr(self, v)
      27 def appendr(self,v):    # append, recursive
      28     if v < 0:
----> 29         raise ValueError("Negative input: " + str(v))
      30     if self.isempty():
      31         self.value = v

ValueError: Negative input: -2

```

```

In [20]: try:
         l = List([1,-2,3])
         except ValueError:
             print("oops")

```

oops

```

In [21]: try:
         l = List([1,-2,3])
         except ValueError as errormsg:    # Saves error value in errormsg
             print(errormsg)

```

Negative input: -2