

Lecture 15, 08 October 2026

Linked lists

- Two versions of `append()`
 - `appendi()`, iterative
 - `appendr()`, recursive
 - Dummy `append()` that calls either `appendi()` or `appendr()`
 - To avoid problems with `__init__`, `__str__`
- Recursive `delete()`
 - Note the check to remove spurious empty node after recursive delete

```
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 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 [2]: 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]
```

```

In [3]: l.delete(13)
        print(l)

```

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

- Some performance measurements

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

```
In [4]: import sys
sys.setrecursionlimit(2**31-1)
```

```
In [5]: import time
```

- Iterative append, quadratic complexity

```
In [6]: 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 0.03675238999858266
2000 0.13731428800019785
3000 0.2546612810001534
4000 0.45674453499668743
```

- Recursive append, also quadratic, but 4x--5x overhead due to recursive calls

```
In [7]: 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 0.15277522699761903
2000 0.574801351998758
3000 1.4010387910020654
4000 2.4859663140014163
```

Exception handling

```
In [8]: print(z)
```

```
-----
-
NameError                                Traceback (most recent call las
t)
Cell In[8], line 1
----> 1 print(z)

NameError: name 'z' is not defined
```

Output

- Can control how `print` works
- `sep` sets how items are separated (default is a single space)
- `end` sets what happens after print (default is a new line)

```
In [9]: x = 5
        y = 7
        print(x,y)
        print(x,y,sep=" ",end=" ")
        print(x,y)
```

```
5 7
5,75 7
```

Input

- Read a line from the keyboard using `input()`
- Input is terminated by the user typing `Enter`
- Returns a string *without* the new line character corresponding to `Enter`

```
In [10]: x = input()
```

```
In [11]: x
```

```
Out[11]: 'a'
```

- Can display a prompt to indicate what is expected

```
In [13]: x = input("Type something:")
```

```
In [14]: x
```

```
Out[14]: 'Hello'
```

- User input is not in our control, can trigger errors

```
In [15]: xstr = input("Enter an integer:")
        x = int(xstr)
        print(x+5)
```

```
-----
-
ValueError                                Traceback (most recent call las
t)
Cell In[15], line 2
      1 xstr = input("Enter an integer:")
----> 2 x = int(xstr)
      3 print(x+5)

ValueError: invalid literal for int() with base 10: 'abc'
```

- Need a way for our code to "trap" and deal with errors --- *exception handling*

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

```
In [16]: invalid = True
while (invalid):
    try:
        xstr = input("Enter a number: ")
        xint = int(xstr)
        invalid = False
    except:
        pass
print(xint)
```

77

- A refined version that prints a diagnostic message to ask the user to retry

```
In [17]: 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:

88

- Catch a specific type of exception

```
In [18]: int("abc")
```

```
-----
-
ValueError                                Traceback (most recent call las
t)
Cell In[18], line 1
----> 1 int("abc")

ValueError: invalid literal for int() with base 10: 'abc'
```

```
In [19]: invalid = True
while (invalid):
    try:
        xstr = input("Enter a number: ")
        xint = int(xstr)
        invalid = False
    except ValueError:
        pass
print(xint)
```

