

Lecture 17, 14 October 2025

- String formatting -- positional arguments

```
In [1]: x = 7
        y = 22
        message = "First one is {1}, second one is {0}"
        message.format(x,y)
```

```
Out[1]: 'First one is 22, second one is 7'
```

```
In [2]: x = 7
        y = 22
        message1 = "First one is {1}, second one is {0}. To repeat, first one is
        message1.format(x,y)
```

```
Out[2]: 'First one is 22, second one is 7. To repeat, first one is 22.'
```

- Raising an exception in `List()`
- Inserting a negative value raises `ValueError`
- Use string formatting to add negative value to error message

```
In [3]: 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:{0}".format(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:{0}".format(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:{0}".format(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 [4]: l = List([1,-2,3])
        print(l)
```

```
-----
-
ValueError                                Traceback (most recent call last)
Cell In[4], line 1
----> 1 l = List([1,-2,3])
      2 print(l)

Cell In[3], line 6, in List.__init__(self, initlist)
      4 self.next = None
      5 for x in initlist:
----> 6     self.append(x)
      7 return

Cell In[3], line 39, in List.append(self, v)
      38 def append(self,v):
----> 39     self.appendr(v)
      40     return

Cell In[3], line 29, in List.appendr(self, v)
      27 def appendr(self,v):    # append, recursive
      28     if v < 0:
----> 29         raise ValueError("Negative input:{0}".format(v))
      30     if self.isempty():
      31         self.value = v

ValueError: Negative input:-2
```

- String formatting with output specification

```
In [5]: "Value {0:8.2f}".format(747.3)
```

```
Out[5]: 'Value    747.30'
```

```
In [6]: "Value {0:6.2f} {0:27.7f}".format(99999999947.3444,22)
```

```
Out[6]: 'Value 99999999947.34          99999999947.3444061'
```

- Modifying behaviour of `print()`

```
In [7]: x = 5
        y = 7
        print(x,y)
        print(y)
```

```
5 7
7
```

```
In [8]: x = 5
        y = 7
        print(x,y,sep=":")
```

5:7

```
In [9]: l = list(range(20))
```

```
In [10]: for x in l:
          print(x,end=" ")
          print("Where are we?")
```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, Where are we?

```
In [11]: for x in l:
          print(x,end=" ")
          print()
          print("Where are we?")
```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, Where are we?

Working with files

Reading files

- Open a file in mode `"r"`

```
In [12]: fh = open("oz.txt","r")
```

- `read()` returns a string with the entire file
- Note the `\n` characters indicating end of line ("new line")

```
In [13]: contents = fh.read()
         contents
```

```
Out[13]: 'I met a traveller from an antique land\nWho said: Two vast and trunkless  
legs of stone\nStand in the desert. Near them, on the sand,\nHalf sunk, a shattered visage lies, whose frown,\nAnd wrinkled lip, and sneer of cold command,\nTell that its sculptor well those passions read\nWhich yet survive, stamped on these lifeless things,\nThe hand that mocked them  
and the heart that fed:\nAnd on the pedestal these words appear:\n"My name is Ozymandias, King of Kings:\nLook on my works, ye Mighty, and despair!"\nNothing beside remains. Round the decay\nOf that colossal wreck, boundless and bare\nThe lone and level sands stretch far away. \n'
```

- After reading the file, we are at the end
- End of file is indicated by `read()` returning the empty string

```
In [14]: fh.read()
```

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

```
In [15]: fh.read()
```

```
Out[15]: ''
```

- To re-read the file, we have to close it and open it again

```
In [16]: fh.close()
```

- `readline()` reads one line
- Unlike `input()`, the line that is read includes the terminating `\n`

```
In [17]: fh = open("oz.txt", "r")
line1 = fh.readline()
line1
```

```
Out[17]: 'I met a traveller from an antique land\n'
```

- `readlines()` returns the entire file as a list of lines
- Each line is terminated by `\n`

```
In [18]: fh.close()
fh = open("oz.txt", "r")
alllines = fh.readlines()
alllines
```

```
Out[18]: ['I met a traveller from an antique land\n',
'Who said: Two vast and trunkless legs of stone\n',
'Stand in the desart. Near them, on the sand,\n',
'Half sunk, a shattered visage lies, whose frown,\n',
'And wrinkled lip, and sneer of cold command,\n',
'Tell that its sculptor well those passions read\n',
'Which yet survive, stamped on these lifeless things,\n',
'The hand that mocked them and the heart that fed:\n',
'And on the pedestal these words appear:\n',
'"My name is Ozymandias, King of Kings:\n',
'Look on my works, ye Mighty, and despair!"\n',
'No thing beside remains. Round the decay\n',
'Of that colossal wreck, boundless and bare\n',
'The lone and level sands stretch far away. \n']
```

- `readlines()` returns the lines from the current position
- In the following example, the list returned is from the second line onwards

```
In [19]: fh.close()
fh = open("oz.txt", "r")
line1 = fh.readline()      # Read first line
alllines = fh.readlines()  # From second line onwards
alllines
```

```
Out[19]: ['Who said: Two vast and trunkless legs of stone\n',
        'Stand in the desaet. Near them, on the sand,\n',
        'Half sunk, a shattered visage lies, whose frown,\n',
        'And wrinkled lip, and sneer of cold command,\n',
        'Tell that its sculptor well those passions read\n',
        'Which yet survive, stamped on these lifeless things,\n',
        'The hand that mocked them and the heart that fed:\n',
        'And on the pedestal these words appear:\n',
        '"My name is Ozymandias, King of Kings:\n',
        'Look on my works, ye Mighty, and despair!"\n',
        'No thing beside remains. Round the decay\n',
        'Of that colossal wreck, boundless and bare\n',
        'The lone and level sands stretch far away. \n']
```

Stripping white space

- We can strip off leading and trailing "white space" (blank, tab, newline) from a string
- `rstrip()` removes trailing whitespace
- Note that strings are immutable, so the function returns a new string, leaving the original untouched

```
In [20]: teststr = "   abc   \n"
        teststr
```

```
Out[20]: '   abc   \n'
```

```
In [21]: teststr.rstrip()
```

```
Out[21]: '   abc'
```

```
In [22]: teststr
```

```
Out[22]: '   abc   \n'
```

- `lstrip()` removes leading whitespace

```
In [23]: teststr.lstrip()
```

```
Out[23]: 'abc   \n'
```

- `strip()` removes whitespace at both ends

```
In [24]: teststr.strip()
```

```
Out[24]: 'abc'
```

- Can use these to quickly remove `\n` from lines that we read from a file
- In the first loop below, there is a blank line between actual lines because of the `\n` in the input line followed by the newline inserted by `print()`

```
In [25]: fh.close()
fh = open("oz.txt", "r")
for l in fh.readlines():
    print(l)
```

I met a traveller from an antique land
Who said: Two vast and trunkless legs of stone
Stand in the desert. Near them, on the sand,
Half sunk, a shattered visage lies, whose frown,
And wrinkled lip, and sneer of cold command,
Tell that its sculptor well those passions read
Which yet survive, stamped on these lifeless things,
The hand that mocked them and the heart that fed:
And on the pedestal these words appear:
"My name is Ozymandias, King of Kings:
Look on my works, ye Mighty, and despair!"
Nothing beside remains. Round the decay
Of that colossal wreck, boundless and bare
The lone and level sands stretch far away.

- If we strip each line before printing, the blank lines are eliminated

```
In [26]: fh.close()
fh = open("oz.txt", "r")
for l in fh.readlines():
    print(l.rstrip())
```

I met a traveller from an antique land
Who said: Two vast and trunkless legs of stone
Stand in the desert. Near them, on the sand,
Half sunk, a shattered visage lies, whose frown,
And wrinkled lip, and sneer of cold command,
Tell that its sculptor well those passions read
Which yet survive, stamped on these lifeless things,
The hand that mocked them and the heart that fed:
And on the pedestal these words appear:
"My name is Ozymandias, King of Kings:
Look on my works, ye Mighty, and despair!"
Nothing beside remains. Round the decay
Of that colossal wreck, boundless and bare
The lone and level sands stretch far away.

Writing files

- Open a file in mode `"w"`
- Opening a non-existent file for reading generates an error
- Opening a non-existent file for writing creates a new file
- If the file already exists, write will overwrite the contents

```
In [27]: infile = open("newfile.txt","r")
```

```
-----
-
FileNotFoundError                                Traceback (most recent call las
t)
Cell In[27], line 1
----> 1 infile = open(          ,      )

File ~/python-venv/lib/python3.13/site-packages/IPython/core/interactivesh
ell.py:343, in _modified_open(file, *args, **kwargs)
    336 if file in {0, 1, 2}:
    337     raise ValueError(
    338         f"IPython won't let you open fd={file} by default "
    339         "as it is likely to crash IPython. If you know what you ar
e doing, "
    340         "you can use builtins' open."
    341     )
--> 343 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'newfile.txt'
```

```
In [28]: outfile = open("newfile.txt","w")
```

```
In [29]: outfile.close()
```

- `fh.write(s)` writes the string `s` to the file associated with file handle `fh`
- Need to add `\n` ourselves to ensure the end of line
- `fh.writelines(sl)` writes a list of strings `sl`
- Again, we need to ensure `\n` is present at the end of each string in the list
 - The name of the function is misleading
 - It is more accurate to call it `writestrings()` since we have to insert `\n` manually
- The following loop copies the input to the output
 - Each line that is read includes the `\n`
 - Hence each line that is written also has the corresponding `\n`
- However, after this loop, typically the output file will be empty
- Need to close the file handle for the buffer to be "flushed" to the disk

```
In [30]: infile = open("oz.txt","r")
outfile = open("newfile.txt","w")
for l in infile.readlines():
    outfile.write(l)
```



```
In [31]: outfile.close()
```

- Close all file handles when you are done with them

```
In [32]: infile = open("oz.txt","r")
outfile = open("newfile.txt","w")
for l in infile.readlines():
    outfile.write(l)
infile.close()
outfile.close()
```

- We can also read all the lines into a list using `readlines()` and write them out using `writelines()`

```
In [33]: infile = open("oz.txt","r")
outfile = open("newfile.txt","w")
contents = infile.readlines()
outfile.writelines(contents)
infile.close()
outfile.close()
```

Appending output

- We can append our writes at the end of a file
- Open the file with mode `"a"` instead of `"w"`
- After the following, `newfile.txt` should have two copies of `oz.txt`

```
In [34]: infile = open("oz.txt","r")
outfile = open("newfile.txt","a")
contents = infile.readlines()
outfile.writelines(contents)
infile.close()
outfile.close()
```

- There are other functions we have not discussed
- `fh.seek(n)` moves to position `n` in the file
 - After reading a file, use `fh.seek(0)` to return to the start
- `fh.read(n)` reads `n` characters from the current position

```
In [35]: infile = open("oz.txt","r")
c1 = infile.read()
infile.seek(0)
c2 = infile.read(20)
```

```
In [36]: c1
```

```
Out[36]: 'I met a traveller from an antique land\nWho said: Two vast and trunkless  
legs of stone\nStand in the desert. Near them, on the sand,\nHalf sunk, a shattered visage lies, whose frown,\nAnd wrinkled lip, and sneer of cold command,\nTell that its sculptor well those passions read\nWhich yet survive, stamped on these lifeless things,\nThe hand that mocked them  
and the heart that fed:\nAnd on the pedestal these words appear:\n\"My name is Ozymandias, King of Kings:\nLook on my works, ye Mighty, and despair!\"\nNothing beside remains. Round the decay\nOf that colossal wreck,  
boundless and bare\nThe lone and level sands stretch far away. \n'
```

```
In [37]: c2
```

```
Out[37]: 'I met a traveller fr'
```