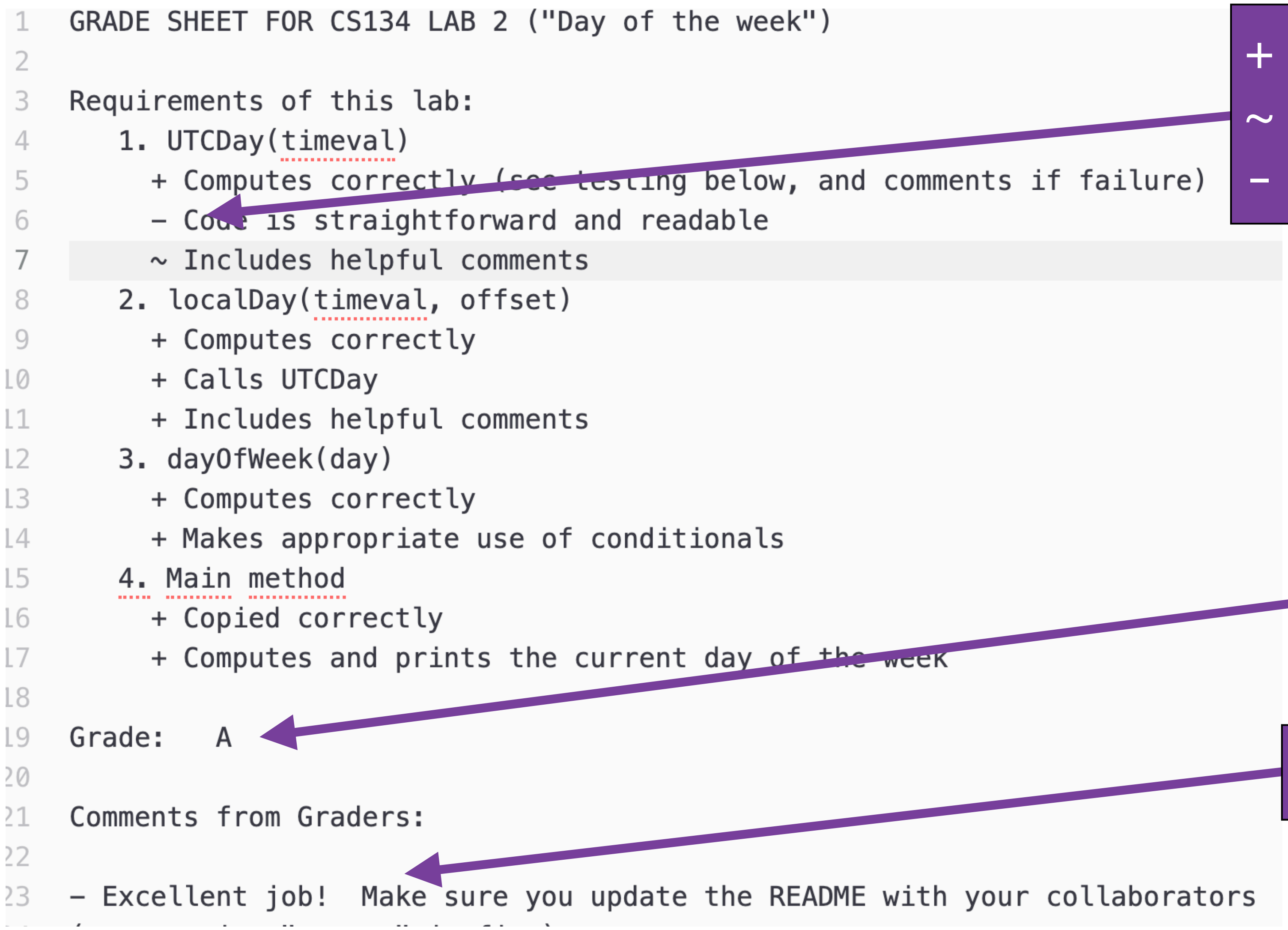
CS134:
Ranges & Files

Announcements & Logistics

- **Homework 4** due next Mon at 10 pm
- **Lab 2** feedback coming soon
- **Lab 3** due today/tomorrow at 10pm
  - Lots of student help hours today/tomorrow if you need help!

Do You Have Any Questions?

Interpreting Lab Feedback



```
GRADE SHEET FOR CS134 LAB 2 ("Day of the week")

Requirements of this lab:
   1. UTCDay(timeval)
     + Computes correctly (see testing below, and comments if failure)
     - Code is straightforward and readable
     ~ Includes helpful comments
   2. localDay(timeval, offset)
     + Computes correctly
     + Calls UTCDay
     + Includes helpful comments
   3. dayOfWeek(day)
     + Computes correctly
     + Makes appropriate use of conditionals
   4. Main method
     + Copied correctly
     + Computes and prints the current day of the week

Grade:   A

Comments from Graders:

- Excellent job!  Make sure you update the README with your collaborators
```

```
+   Good
~   Okay
-   Needs work
```

Grade

Comments

Look for comments that start with #$ in your code.

Last Time

- Learned about **nested for loops**
- Summarized important **string and list methods and operations**
  - Sequence operators and functions: `+, [], [:], *,` etc
    - All sequence ops work similarly on strings and lists
    - None of them change the original string or list
  - String methods: `.lower(), .upper(), .join(), .split()`
  - List methods: `.append(), .extend()`

Today's Plan

- Review adding items to lists using **+**, **append()**, and **extend()**
  - Begin thinking about side effects of mutability in lists
- Discuss **ranges**: as an easy way to generate numerical sequences
- Discuss **file reading and writing** using lists and strings (like **readWords()** from lab)
- We'll return to more advanced list functionality on Friday

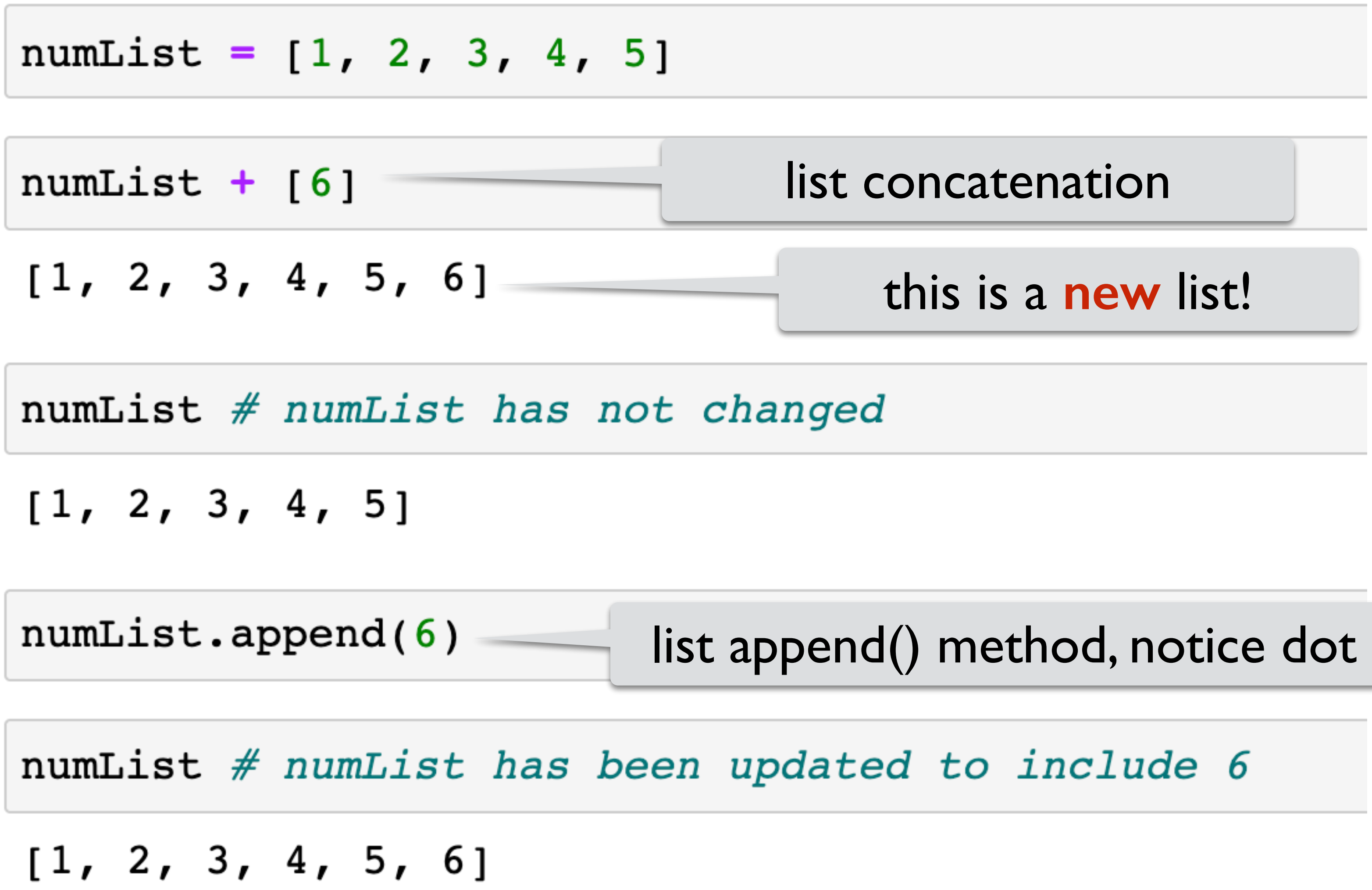
Recap: Modifying Lists

- Unlike strings, lists are **mutable** data structures
  - We can *change* them (delete things from them, add things to them, etc.)
- List **concatenation** (using **+**) ***creates a new list*** and ***does not modify (or mutate)*** any existing list
  - **Important point: Concatenating to a list using + returns a new list!**

- Alternatively we can **append to a list** using a special list method
  - The list **method** `myList.append(item)` ***modifies*** the list `myList` by adding `item` to it at the end
  - Often more efficient to append rather than concatenate! (But we have to be very careful when modifying the list)
  - **Important point: Appending to a list modifies the existing list!**

Adding elements to a List

- Here are a few examples that show how to use the list `.append()` method vs **+** operator to add items to the end of an existing list



```
numList = [1, 2, 3, 4, 5]
```

```
numList + [6]
```

list concatenation

```
[1, 2, 3, 4, 5, 6]
```

this is a **new** list!

```
numList # numList has not changed
```

```
[1, 2, 3, 4, 5]
```

```
numList.append(6)
```

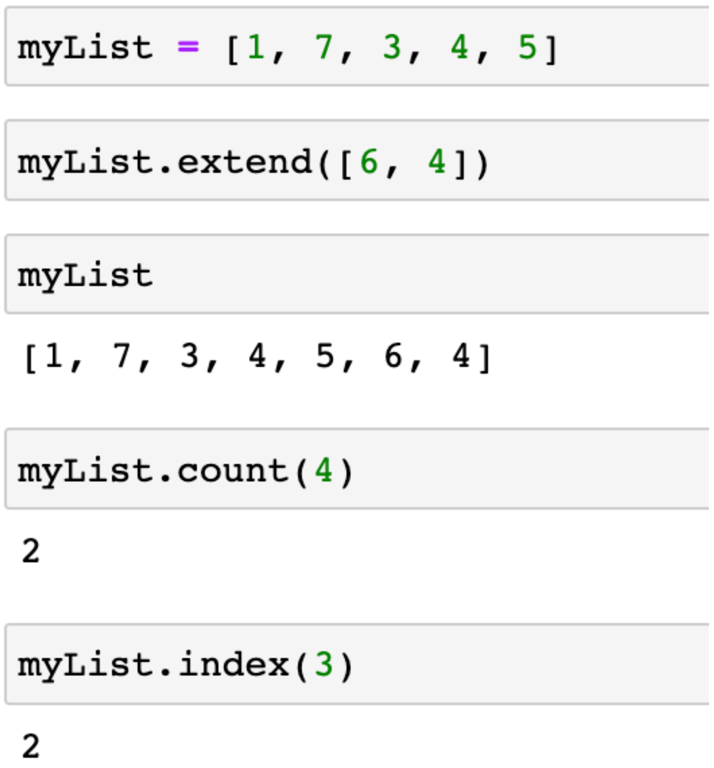
list append() method, notice dot notation

```
numList # numList has been updated to include 6
```

```
[1, 2, 3, 4, 5, 6]
```

More Useful List Methods

- `myList.extend(itemList)`: ***appends all items*** in `itemList` to the end of `myList` (modifying `myList`)
- `myList.count(item)`: counts and returns the number (`int`) of times `item` appears in `myList`
- `myList.index(item)`: returns the first index (`int`) of item in myList if it is present, else throws an error



```
myList = [1, 7, 3, 4, 5]
```

```
myList.extend([6, 4])
```

```
myList
```

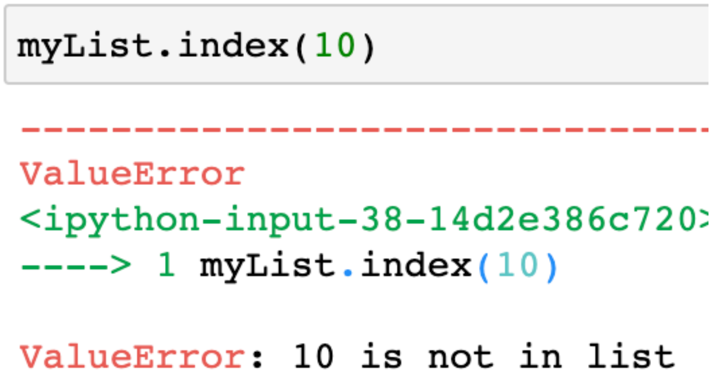
```
[1, 7, 3, 4, 5, 6, 4]
```

```
myList.count(4)
```

```
2
```

```
myList.index(3)
```

```
2
```



```
myList.index(10)
```

```
---------------------------------
ValueError
<ipython-input-38-14d2e386c720
----> 1 myList.index(10)

ValueError: 10 is not in list
```

Summarizing Mutability in Strings vs Lists

Strings are immutable

- Once you create a string, it cannot be changed!
- All operations that we have seen on strings ***return a new string*** and ***do not modify*** the original string

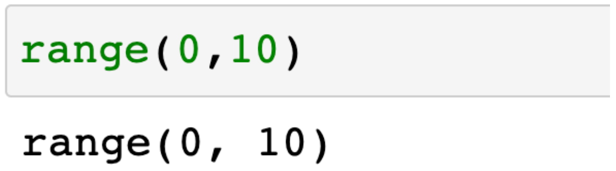
Lists are mutable

- Lists are mutable (or changeable) sequences
- We concatenate items to a list using **+**, but this ***does not*** change the list
- We append items using **append()** method, and this ***does*** change the list
- Next week we'll revisit list mutability in more detail!

Ranges

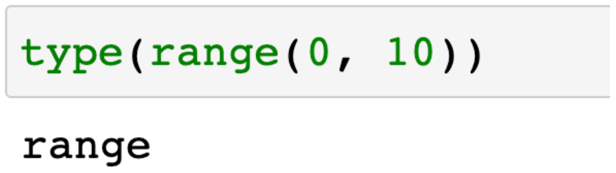
Moving on: Ranges (another sequence!)

- Python provides an easy way to iterate over numerical sequences using the **range** data type, **another sequence**
- When the `range()` function is given two integer arguments, it returns a ***range object*** of all integers starting at the first and up to, *but not including*, the second; note: default starting value is 0
- To see the values included in the range, we can pass our range to the `list()` function which returns a **list** of them



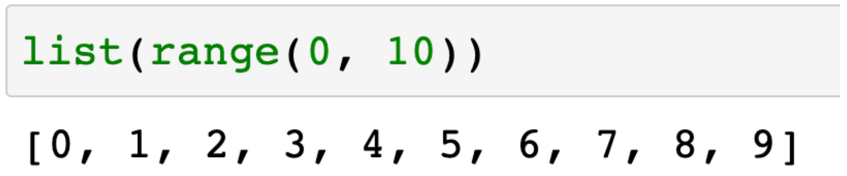
```
range(0,10)
```

```
range(0, 10)
```



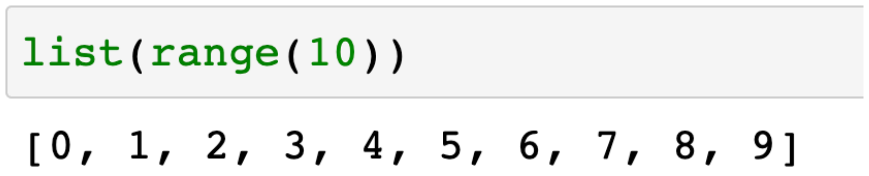
```
type(range(0, 10))
```

```
range
```



```
list(range(0, 10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Moving on: Ranges (another sequence!)

- Python provides an easy way to iterate over numerical sequences using the **range** data type, **another sequence**
- When the `range()` function is given two integer arguments, it returns a ***range object*** of all integers starting at the first and up to, *but not including*, the second; note: default starting value is 0
- To see the values included in the range, we can pass our range to the `list()` **list** of them

A range is a type of sequence in Python (like string and list)

```
range(0,10)
```

```
range(0, 10)
```

```
type(range(0, 10))
```

```
range
```

To see elements in range, pass range to list() function

```
list(range(0, 10))
```

```
[0, 1, 2, 3, 4,
```

First argument omitted, defaults to 0

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Loops and Ranges to Print Patterns

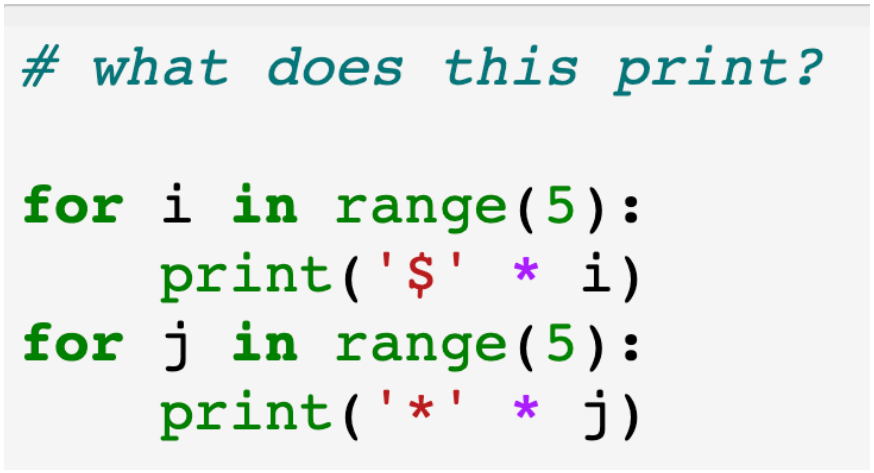
- In addition to iterating over strings and lists, we can use a **for loop** and a range to simply **repeat** a task. The following loops print a pattern to the screen. (Look closely at the indentation!)
-

```python
# what does this print?

for i in range(5):
    print('$' * i)
for j in range(5):
    print('*' * j)
```

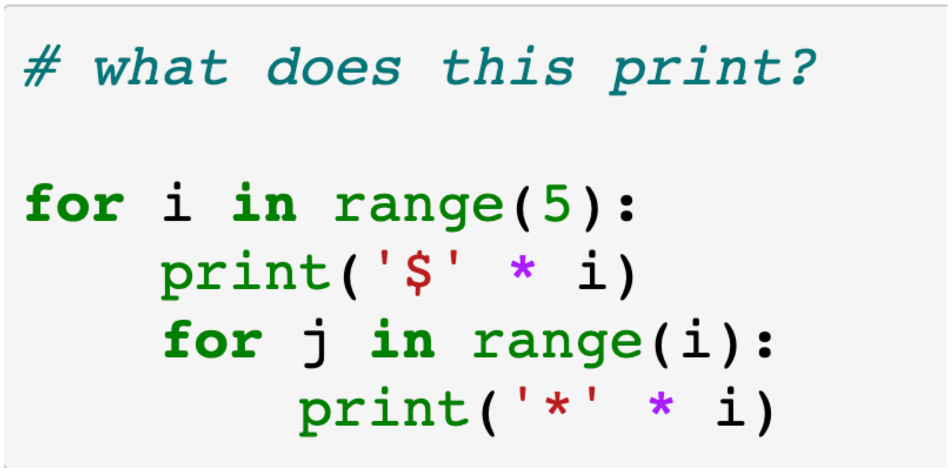
```python
# what does this print?

for i in range(5):
    print('$' * i)
    for j in range(i):
        print('*' * i)
```

What are the values of i and j???

Iterating Over Ranges



```python
# what does this print?

for i in range(5):
    print('$' * i)
for j in range(5):
    print('*' * j)
```



```python
# what does this print?

for i in range(5):
    print('$' * i)
    for j in range(i):
        print('*' * i)
```

Iterating Over Ranges

```python
# what does this print?

for i in range(5):
    print('$' * i)
for j in range(5):
    print('*' * j)
```

```
              i = 0
$             i = 1
$$            i = 2
$$$           i = 3
$$$$          i = 4
              j = 0
*             j = 1
**            j = 2
***           j = 3
****          j = 4
```

```python
# what does this print?

for i in range(5):
    print('$' * i)
    for j in range(i):
        print('*' * i)
```

i, not j!

```
              i = 0
$             i = 1
                    j = 0
*
$$            i = 2
                    j = 0
**
                    j = 1
**
$$$           i = 3
                    j = 0
***
                    j = 1
***
                    j = 2
***
$$$$          i = 4
****                j = 0
****                j = 1
****                j = 2
****                j = 3
```

Reading Data from Files

Working with Files in Python

- File I/O is a very common and important operation
- `open(filename, mode)` is a built-in Python function for working with files
  - `filename` is a path to a file as a **string**
  - `mode` is a string where
    - `'r'` - open for reading (default)
    - `'w'` - open for writing (will overwrite previous contents)
    - `'a'` - open for appending (will not overwrite previous contents)

- Using `open()` within a `with … as` code block, we can **iterate** over the **lines of a text file** just as we iterated over strings and lists in previous lectures

Opening Files: `with … as`

Path to file on computer as a string

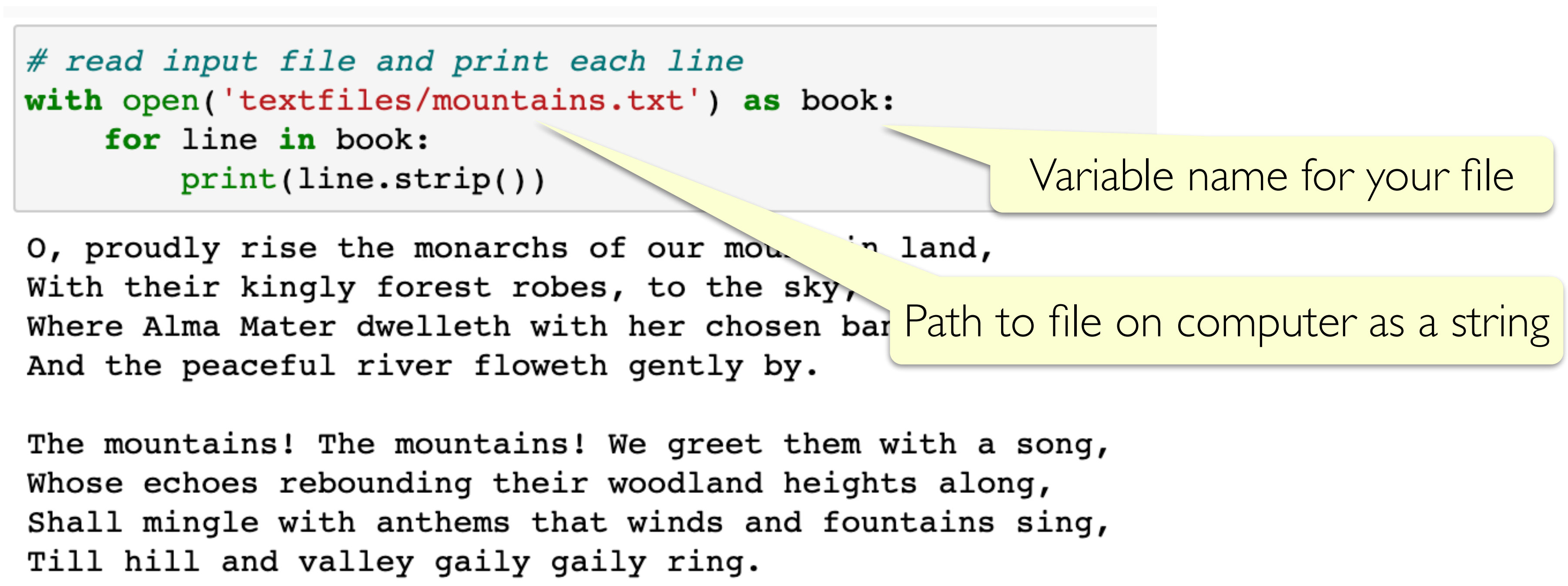
```
with open(filename) as inputFile:
    # do something with file
```

Variable name for your file

Note. **(syntax)** Indentation defines the body of the with block where the file is open. File automatically closed after with…as block.

Iterating over Lines in a File

- Within a `with open(filename) as inputFile:` block, we can iterate over the lines in the file just as we would iterate over any sequence such as lists, strings, or ranges
- The end of a line in the text file is determined by the special newline character `'\n'`
- Example: We have a text file `mountains.txt` within a directory `textfiles`. We can iterate and print each line as follows:



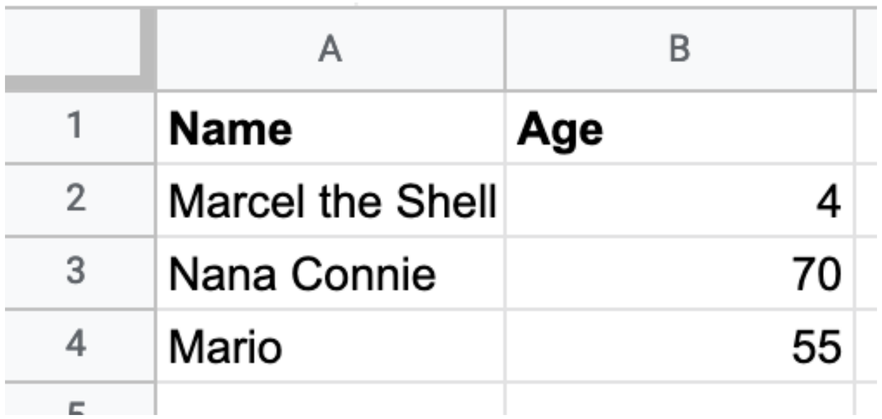
```python
# read input file and print each line
with open('textfiles/mountains.txt') as book:
    for line in book:
        print(line.strip())
```

```
O, proudly rise the monarchs of our mou    n land,
With their kingly forest robes, to the sky,
Where Alma Mater dwelleth with her chosen bar
And the peaceful river floweth gently by.

The mountains! The mountains! We greet them with a song,
Whose echoes rebounding their woodland heights along,
Shall mingle with anthems that winds and fountains sing,
Till hill and valley gaily gaily ring.
```

Variable name for your file
Path to file on computer as a string

Common File Type: CSVs

- A CSV (Comma Separated Values) file is a specific type of plain text file that stores "tabular" data
- Each row of a table is a line in the text file, with each column on the row separated by commas
- This format is a common import and export format for spreadsheets and databases



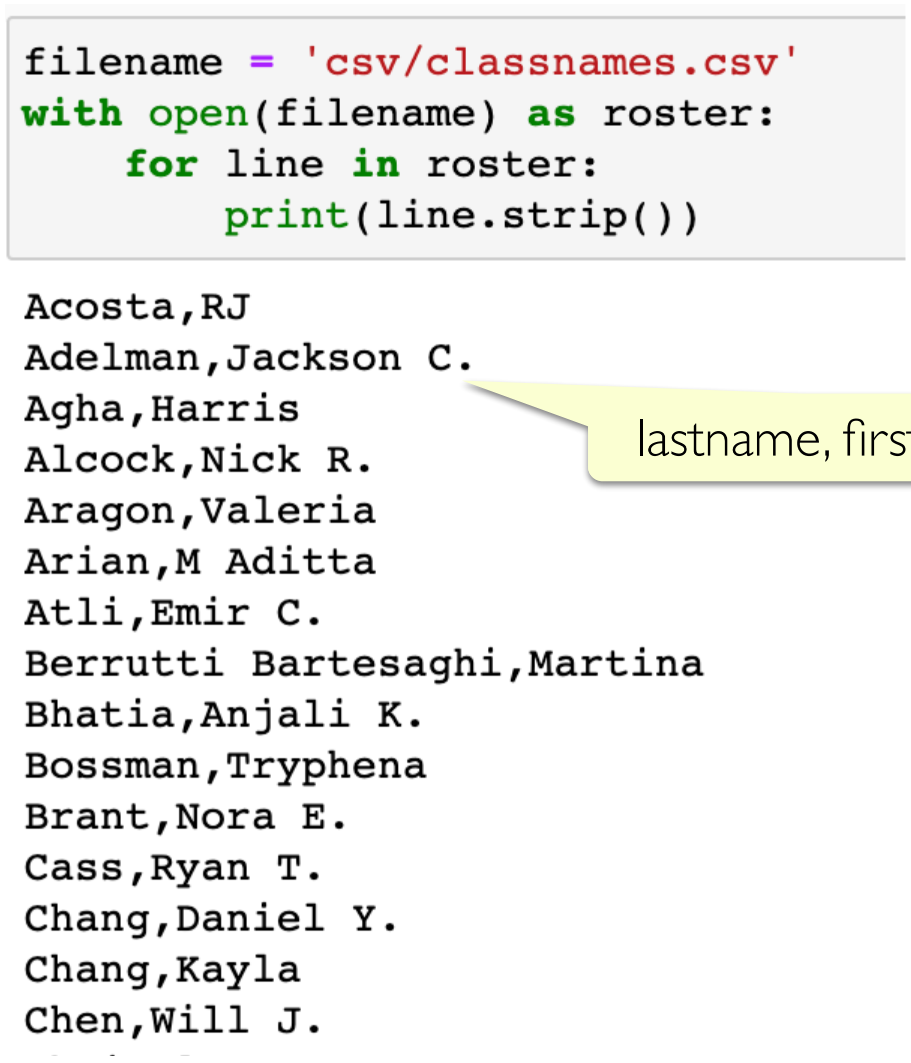
	A	B
1	**Name**	**Age**
2	Marcel the Shell	4
3	Nana Connie	70
4	Mario	55
5		

CSV form:

```
Name,Age
Marcel the Shell,4
Nana Connie,70
Mario,55
```

Working with CSVs

- Since CSVs are just text files, we can process them in the same way
- Might require additional post-processing/splitting using string methods



```python
filename = 'csv/classnames.csv'
with open(filename) as roster:
    for line in roster:
        print(line.strip())
```

```
Acosta,RJ
Adelman,Jackson C.
Agha,Harris
Alcock,Nick R.
Aragon,Valeria
Arian,M Aditta
Atli,Emir C.
Berrutti Bartesaghi,Martina
Bhatia,Anjali K.
Bossman,Tryphena
Brant,Nora E.
Cass,Ryan T.
Chang,Daniel Y.
Chang,Kayla
Chen,Will J.
```

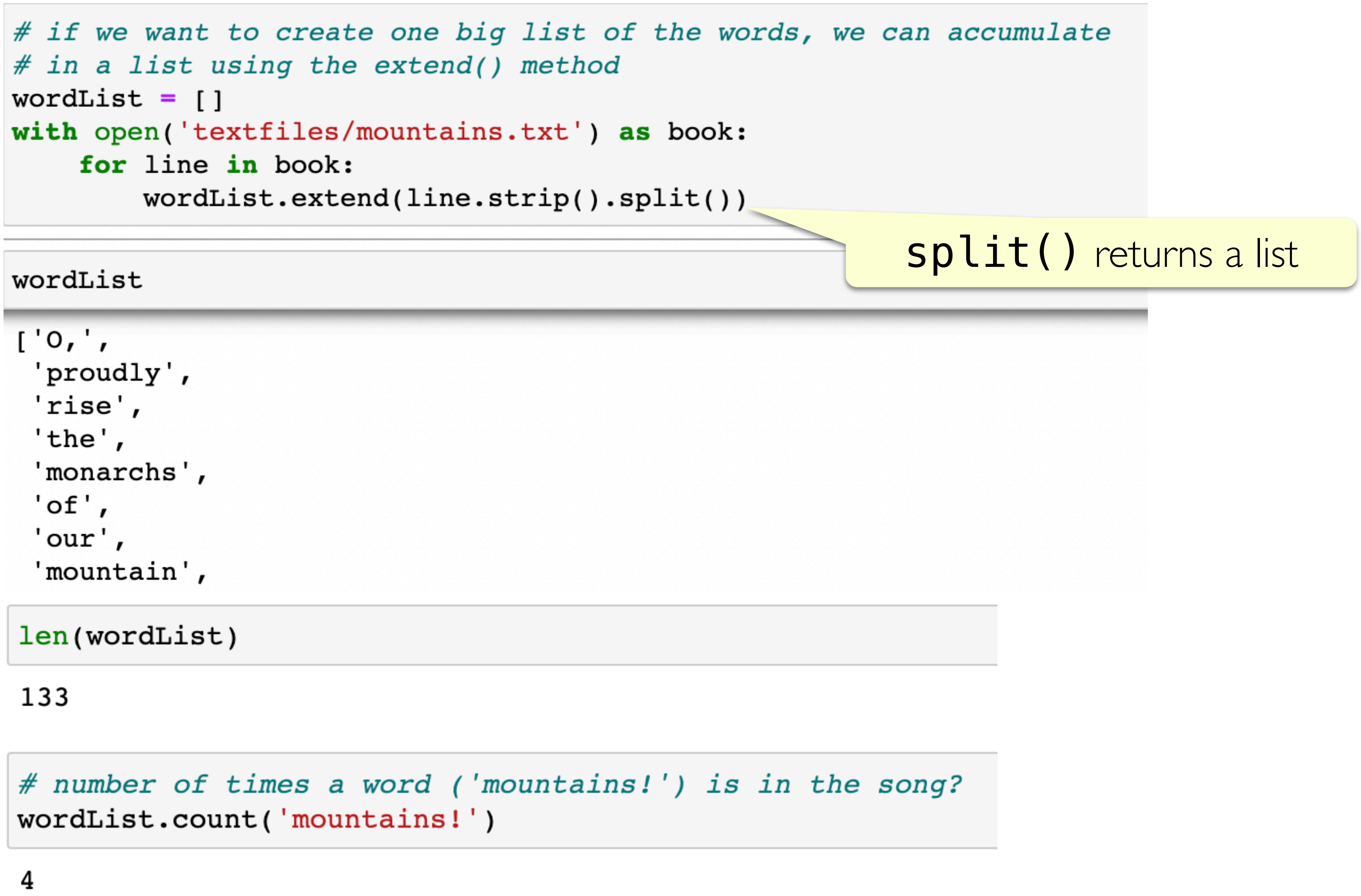
lastname, firstname

Useful String and List Methods in File Reading

- When reading files, we can use our favorite list and string methods to work with the data
  - `line.strip():` Remove any leading/trailing white space or "\n"
  - `line.split(','):` Separate a **comma-separated** sequence of words and create a list of strings
  - `' '.join(line.split(',')):` Create a single "big" string with words separated by spaces instead of commas
  - `myList.extend():` Create lists of words while iterating over the file
  - `myList.count(ele):` Count the occurrence of various elements
  - …and so on!

Data Analysis

- Some examples (more on Jupyter!)



```
# if we want to create one big list of the words, we can accumulate
# in a list using the extend() method
wordList = []
with open('textfiles/mountains.txt') as book:
    for line in book:
        wordList.extend(line.strip().split())
```

split() returns a list

```
wordList
```

```
['O,',
 'proudly',
 'rise',
 'the',
 'monarchs',
 'of',
 'our',
 'mountain',
```

```
len(wordList)
```

```
133
```

```
# number of times a word ('mountains!') is in the song?
wordList.count('mountains!')
```

```
4
```

Data Analysis w/ CSVs

- Convert our last, first CSV (snippet shown below) into a list of names

```
Acosta,RJ
Adelman,Jackson C.
Agha,Harris
Alcock,Nick R.
```

lastname, firstname (CSV)

```python
students = [] # initialize empty list
filename = "csv/classNames.csv"
with open(filename) as roster:
    for line in roster:
        fullName = line.strip().split(',')
        firstName = fullName[1]
        lastName = fullName[0]
        # print(firstName,lastName)
        students.append(firstName + ' ' + lastName)
```

string parsing to find first and last names; then append string to list

```python
students
```

```
['RJ Acosta',
 'Jackson C. Adelman',
 'Harris Agha',
 'Nick R. Alcock',
 'Valeria Aragon',
 'M Aditta Arian',
 'Emir C. Atli',
 'Martina Berrutti Bartesaghi',
 'Anjali K. Bhatia',
```

Final result: a list of strings

Writing to Files

Writing to Files

- In addition to reading, we can also ***write to*** files
- Example: Write all student names into a file.
- To open a ***new*** file for writing, we use `open` with the mode '`w`'.
- Use `.write()` file method to add a string to a file

```
with open('studentNames.txt', 'w') as sFile:
    sFile.write('CS134 students:\n') # need newlines
    sFile.write('\n'.join(students))
```

convert student list into a string separated by new lines

Appending to Files

- If a file already has something in it, opening it in `w` mode again will erase all of its past contents
- Instead we can ***append*** something to an ***existing*** file without erasing the contents. To do that we open in append `a` mode.

```python
with open('studentNames.txt', 'a') as sFile:
    sFile.write('\nGoodbye.\n')
```

```
cat studentNames.txt
```

```
Winnie Zhang
Nicole S. Zhou
Addison Zou
Goodbye.
```

This is the end of our studentNames file

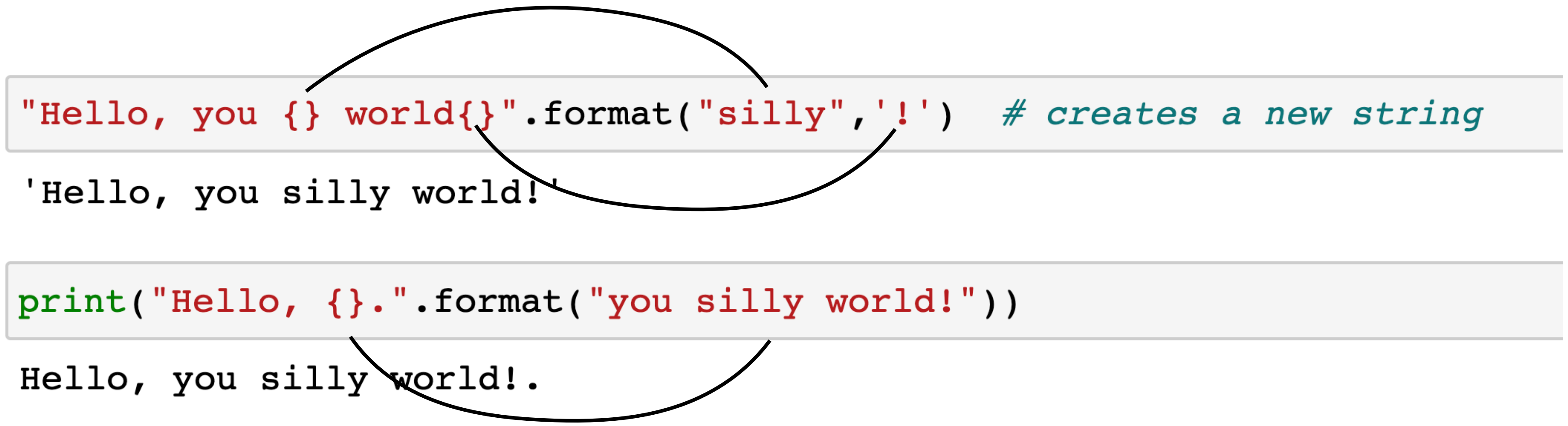
An Aside: Format Printing for Strings

- A convenient way to build strings with particular form is to use the `.format()` string method (you've seen this in lab)

  Syntax: `myString.format(*args)`

  `*args` means it takes zero or more arguments

- For every pair of braces (`{}`), format ***consumes*** one argument
- Argument is ***implicitly converted to a string*** and concatenated with the remaining parts of the format string
- Especially useful when writing to files



```
"Hello, you {} world{}".format("silly",'!')  # creates a new string
'Hello, you silly world!'
```

```
print("Hello, {}.".format("you silly world!"))
Hello, you silly world!.
```

The end!

CS134:

Ranges & Files