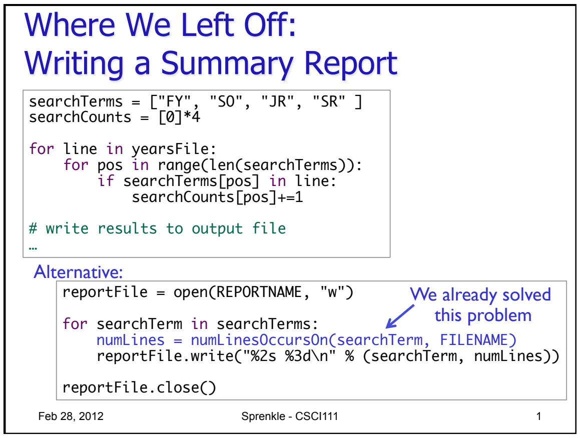


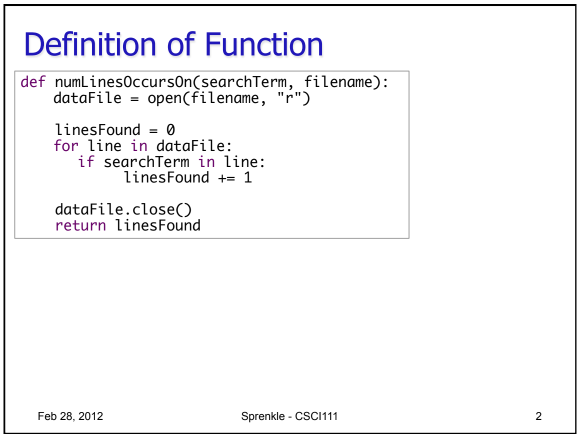
Where We Left Off: Writing a Summary Report

```
searchTerms = ["FY", "SO", "JR", "SR" ]
searchCounts = [0]*4

for line in yearsFile:
    for pos in range(len(searchTerms)):
        if searchTerms[pos] in line:
            searchCounts[pos]+=1

# write results to output file
…
```

Alternative:

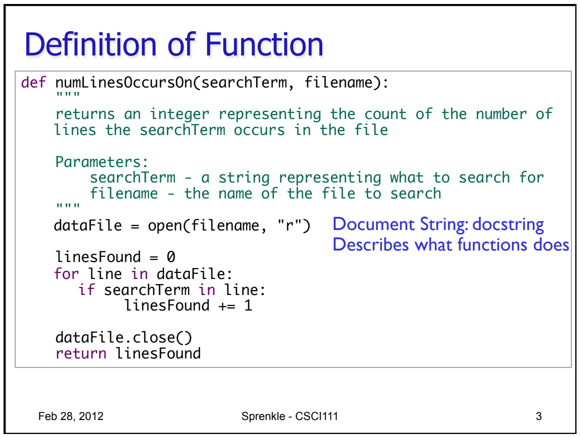
```
reportFile = open(REPORTNAME, "w")

for searchTerm in searchTerms:
    numLines = numLinesOccursOn(searchTerm, FILENAME)
    reportFile.write("%2s %3d\n" % (searchTerm, numLines))

reportFile.close()
```

We already solved this problem



Definition of Function

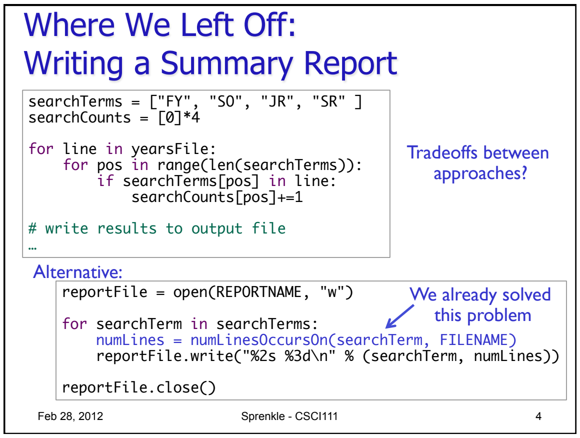
```
def numLinesOccursOn(searchTerm, filename):
    dataFile = open(filename, "r")

    linesFound = 0
    for line in dataFile:
       if searchTerm in line:
             linesFound += 1

    dataFile.close()
    return linesFound
```



Definition of Function

```
def numLinesOccursOn(searchTerm, filename):
    """
    returns an integer representing the count of the number of
    lines the searchTerm occurs in the file

    Parameters:
        searchTerm - a string representing what to search for
        filename - the name of the file to search
    """
    dataFile = open(filename, "r")

    linesFound = 0
    for line in dataFile:
       if searchTerm in line:
             linesFound += 1

    dataFile.close()
    return linesFound
```

Document String: docstring
Describes what functions does



Where We Left Off: Writing a Summary Report

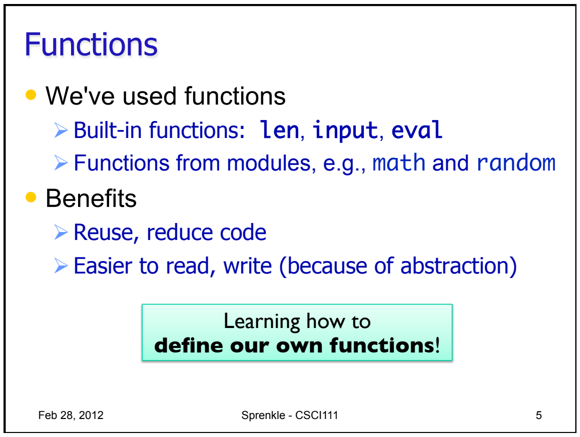
```
searchTerms = ["FY", "SO", "JR", "SR" ]
searchCounts = [0]*4

for line in yearsFile:
    for pos in range(len(searchTerms)):
        if searchTerms[pos] in line:
            searchCounts[pos]+=1

# write results to output file
…
```

Tradeoffs between approaches?

Alternative:

```
reportFile = open(REPORTNAME, "w")

for searchTerm in searchTerms:
    numLines = numLinesOccursOn(searchTerm, FILENAME)
    reportFile.write("%2s %3d\n" % (searchTerm, numLines))

reportFile.close()
```

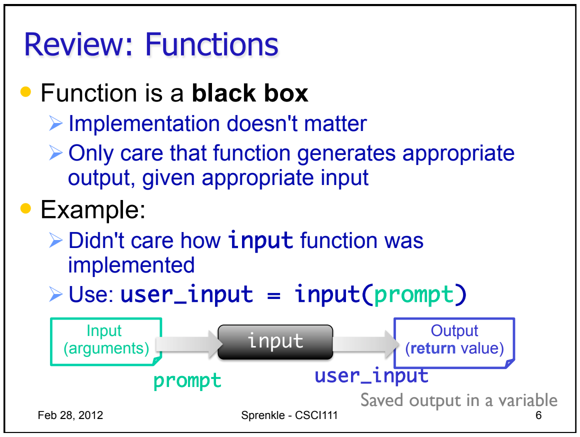
We already solved this problem



Functions

- We've used functions
  - Built-in functions: `len`, `input`, `eval`
  - Functions from modules, e.g., `math` and `random`
- Benefits
  - Reuse, reduce code
  - Easier to read, write (because of abstraction)

Learning how to
define our own functions!



Review: Functions

- Function is a **black box**
  - Implementation doesn't matter
  - Only care that function generates appropriate output, given appropriate input
- Example:
  - Didn't care how `input` function was implemented
  - Use: `user_input = input(prompt)`

Input (arguments) → input → Output (**return** value)

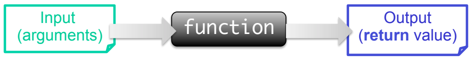
prompt

user_input

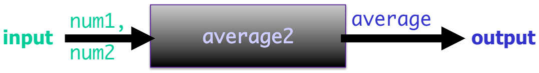
Saved output in a variable

Creating Functions

- A function can have
  - 0 or more inputs
  - 0 or 1 outputs
- When we define a function, we know its inputs and if it has output

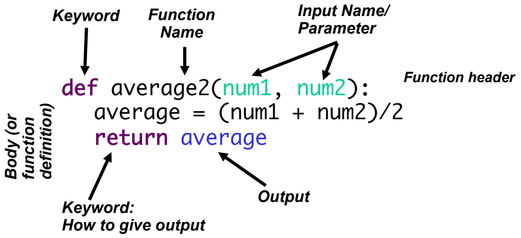


Averaging Two Numbers



- Input: the two numbers
- Output: the average of two numbers

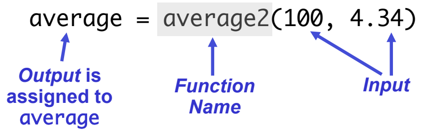
Syntax of Function Definition



```
def average2(num1, num2):
    average = (num1 + num2)/2
    return average
```

Calling your own functions

Same as calling someone else's functions …



```
average = average2(100, 4.34)
```

average2.py

All Together

We will talk more about program organization and how the program executes. For today, know that calling a function defined in your program is similar to calling a function defined in another module.

```
def average2(num1, num2):
    """
    returns the average of two numbers
    Parameters: two numbers
    """
    average = (num1 + num2)/2
    return average

print("This program will find the average of two numbers.")
print()

num1 = eval(input("Enter the first number: " ))
num2 = eval(input("Enter the second number: "))

# call the average function
average = average2(num1, num2)

print("The average of", num1, "and", num2, "is", average)
```

average2.py

Lab 5

- What do I expect from programs?

Lab 5 Feedback

- Big stride in program difficulty
  - Continue to make big strides
- Some techniques should be automatic
  - Good variable names (only constants all caps)
  - Getting user input
  - Validating user input
  - Good test cases
  - Descriptive output
  - Efficient if statements
    - Use elif/else as appropriate
  - When need accumulator

Example: For problem with finding alphabetically first of three words, what should the test cases be?

Lab 5 Feedback

- Test cases for alphabetical first of three strings
  - At least one in each position
- Efficiency of finding the alphabetical first of three strings
  - A few people wrote an efficient solution
  - Later this semester, we'll discuss the *most efficient* solution

Most Common Error

- Doesn't always compute the correct first word

```
if word1 < word2 and word1 < word3:
    print(“The alphabetically first word is”, word1)
elif word2 < word1 and word2 < word3:
    print(“The alphabetically first word is”, word2)
else:
    print(“The alphabetically first word is”, word3)
```

- Where is the problem?

Lab 5 Feedback

- Refactor to type output only once
  - Separate display from computation

```
if word1 <= word2 and word1 <= word3:
    print(“The alphabetically first word is”, word1)
elif word2 <= word1 and word2 <= word3:
    print(“The alphabetically first word is”, word2)
else:
    print(“The alphabetically first word is”, word3)
print(“The alphabetically first word is”, first)
```

Vs.

```
if word1 <= word2 and word1 <= word3:
    first = word1
elif word2 <= word1 and word2 <= word3:
    first = word2
else:
    first = word3
print(“The alphabetically first word is”, first)
```

Which Loop to Use?

- Prob 3: Reversing String

```
for index in range(len(userString)-1,-1,-1):
    backwardsString += userString[index]
```

vs.

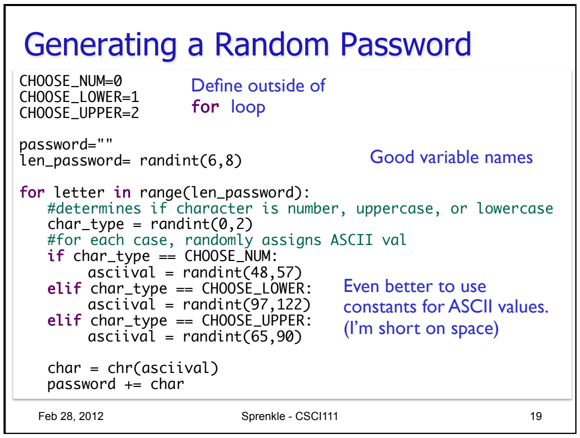
```
for character in userString:
    revString = character + revString
```

If you have a different solution, it's probably too complicated.

Palindrome

- Keep it simple
  - Remove spaces and lower case original word before reversing
    - Don't have to do operations twice
- More efficient solution

```
def isPalindrome(phrase):
    for i in range(len(phrase)/2):
        if phrase[i] != phrase[-i-1]:
            return False
    return True
```



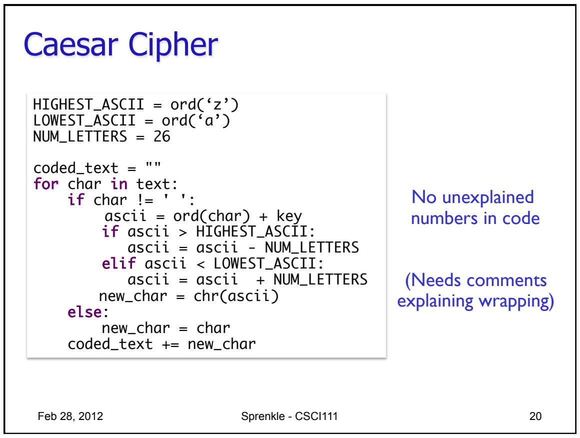
Generating a Random Password

```
CHOOSE_NUM=0
CHOOSE_LOWER=1
CHOOSE_UPPER=2

password=""
len_password= randint(6,8)

for letter in range(len_password):
   #determines if character is number, uppercase, or lowercase
   char_type = randint(0,2)
   #for each case, randomly assigns ASCII val
   if char_type == CHOOSE_NUM:
        asciival = randint(48,57)
   elif char_type == CHOOSE_LOWER:
        asciival = randint(97,122)
   elif char_type == CHOOSE_UPPER:
        asciival = randint(65,90)

   char = chr(asciival)
   password += char
```

Define outside of `for` loop

Good variable names

Even better to use constants for ASCII values. (I'm short on space)

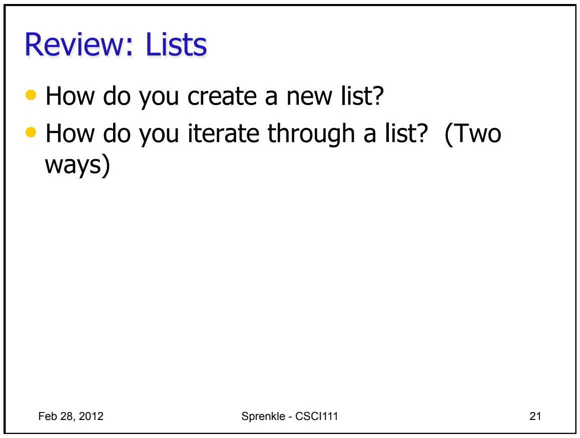


Caesar Cipher

```
HIGHEST_ASCII = ord('z')
LOWEST_ASCII = ord('a')
NUM_LETTERS = 26

coded_text = ""
for char in text:
    if char != ' ':
        ascii = ord(char) + key
        if ascii > HIGHEST_ASCII:
           ascii = ascii - NUM_LETTERS
        elif ascii < LOWEST_ASCII:
           ascii = ascii  + NUM_LETTERS
        new_char = chr(ascii)
    else:
        new_char = char
    coded_text += new_char
```

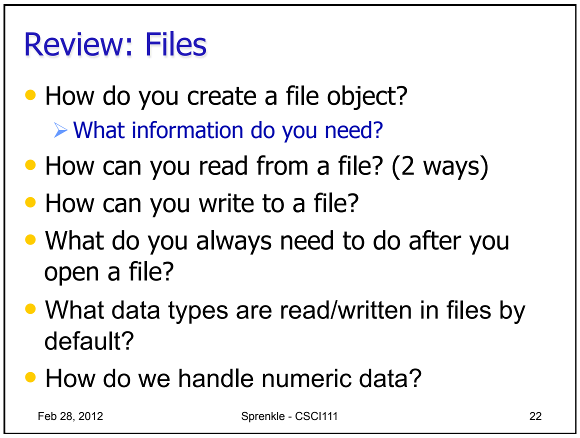
No unexplained numbers in code

(Needs comments explaining wrapping)



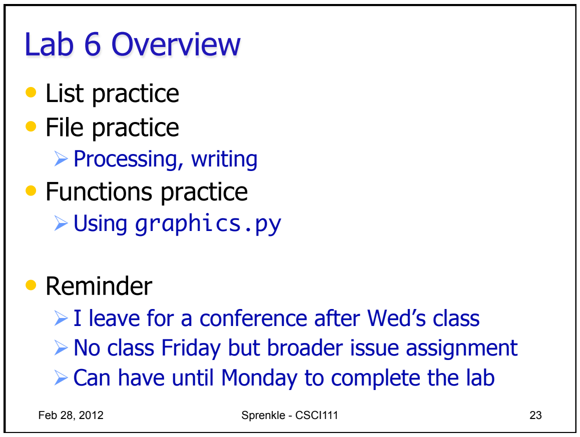
Review: Lists

- How do you create a new list?
- How do you iterate through a list? (Two ways)



Review: Files

- How do you create a file object?
  - What information do you need?
- How can you read from a file? (2 ways)
- How can you write to a file?
- What do you always need to do after you open a file?
- What data types are read/written in files by default?
- How do we handle numeric data?



Lab 6 Overview

- List practice
- File practice
  - Processing, writing
- Functions practice
  - Using `graphics.py`
- Reminder
  - I leave for a conference after Wed's class
  - No class Friday but broader issue assignment
  - Can have until Monday to complete the lab