Lecture 7: Midway Recap and Midterm Prep

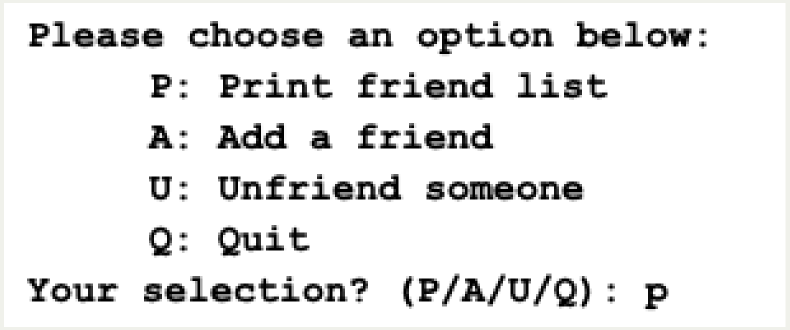
CS5001 / CS5003:
Intensive Foundations
of Computer Science

PDF of this presentation

Lecture 6: Lab Review

- Let's first talk about last week's lab on Facebook and the Seven Dwarfs

I didn't mean for you to write the **`get_choice()`** function -- oops! But, here is the sample output, and it should be straightforward to figure out how to write the function:



```
Please choose an option below:
       P: Print friend list
       A: Add a friend
       U: Unfriend someone
       Q: Quit
Your selection? (P/A/U/Q): p
```

```python
def get_choice():
    """
    Ask the user to make a selection between Print friend list, Add a friend,
      Unfriend someone, and Quit
    :return: 'p', 'a', 'u', or 'q'
    """
```

Lecture 6: Lab Review

- Let's first talk about last week's lab on Facebook and the Seven Dwarfs

```python
def get_choice():
    """
    Ask the user to make a selection between Print friend list, Add a friend,
      Unfriend someone, and Quit
    :return: 'p', 'a', 'u', or 'q'
    """
    print("Please choose an option below:")
    print("\tP: Print a friend list")
    print("\tA: Add a friend")
    print("\tU: Unfriend Someone")
    print("\tQ: Quit")
    choice = ''
    while choice not in ['p', 'a', 'u', 'q']:
        choice = input("Your selection? (P/A/U/Q): ").lower()
    return choice
```

- The `'\t'` means "tab" (you could have put a couple of spaces)
- It is nice to let the user put upper- or lowercase letters, so we just convert the choice to lowercase using the string's `.lower()` function.

Lecture 6: Lab Review

- Here is the **`read_friends()`** docstring:

```python
def read_friends(filename):
    """
    Reads in a Dwarf Facebook list from filename. Returns a dict,
    with each dwarf as the keys, and a list of dwarfs as the value,
    representing the dwarf's friends.
    The file should be formatted as follows:

    dwarf_name1 friend1 friend2 friend3 ...
    dwarf_name2 friend1 friend2 friend3 ...
    ...

    The names will be space-separated.

    A particular Dwarf's dict key/value might look like this:
    dwarf_friends = {
        'Happy': ['Dopey', 'Bashful', 'Sneezy', 'Sleepy', 'Doc', 'Grumpy'],
    }

    :param filename: The file name of the file to read from
    :return: A dict of dwarf friends
    """
    database = {}
    try:
        with open(filename, "r") as f:
            pass
        return database
    except FileNotFoundError:
        print(f"Could not open the database file, '{filename}'")
        return None
```

We need to read in the space separated words, with the first as the key and the rest as the friends.

Lecture 6: Lab Review

```python
def read_friends(filename):
    database = {}
    try:
        with open(filename, "r") as f:
            for line in f:
                line = line[:-1]  # remove newline
                dwarfs = line.split(' ')  # space separated
                database[dwarfs[0]] = dwarfs[1:]  # add first as key, rest as friends
        return database
    except FileNotFoundError:
        print(f"Could not open the database file, '{filename}'")
        return None
```

- Use slices when you can!
- Use the string's `.split()` method when you have data that is separated by a particular character
- Know how to add to a dictionary:
  - `d[key] = value`

Lecture 6: Lab Review

- The `print_friends()` function:

```python
def print_friends(friend_database, dwarf):
    """
    Prints out the friends for a particular dwarf from friend_database. The list
    should print as follows, for example:
    Happy, your friends are:
    Dopey
    Bashful
    Sneezy
    Sleepy
    Doc
    Grumpy

    :param friend_database The database of friends
    :param dwarf: The dwarf whose friends will get printed
    :return: True if the dwarf is in the database, False otherwise
    """
    pass
```

Lecture 6: Lab Review

- The `print_friends()` function:

```python
def print_friends(friend_database, dwarf):
    """
    Prints out the friends for a particular dwarf from friend_database. The list
    should print as follows, for example:
    Happy, your friends are:
    Dopey
    Bashful
    Sneezy
    Sleepy
    Doc
    Grumpy

    :param friend_database The database of friends
    :param dwarf: The dwarf whose friends will get printed
    :return: True if the dwarf is in the database, False otherwise
    """
    print(f"{dwarf}, your friends are:")
    friend_list = friend_database[dwarf] # get friend list from database
    for friend in friend_list:
        print(friend)
```

Lecture 6: Lab Review

- The **`add_friend()`** function:

```python
def add_friend(friend_database, dwarf, new_friend):
    """
    Adds new_friend to the dwarf's friend list, but only if the friend is not
    already a friend
    :param friend_database:
    :param dwarf: The dwarf who is adding a new friend
    :param new_friend: The new friend to add
    :return: True if the friend was added, False otherwise
    """
    pass
```

Lecture 6: Lab Review

- The **`add_friend()`** function:

```
 1 def add_friend(friend_database, dwarf, new_friend):
 2     """
 3     Adds new_friend to the dwarf's friend list, but only if the friend is not
 4     already a friend
 5     :param friend_database:
 6     :param dwarf: The dwarf who is adding a new friend
 7     :param new_friend: The new friend to add
 8     :return: True if the friend was added, False otherwise
 9     """
10     friend_list = friend_database[dwarf] # get friend list from database
11     if new_friend in friend_list:
12         print(f"{new_friend} is already in your friend list. No change.made!")
13         return False
14     else:
15         friend_list.append(new_friend)
16         return True
```

- Don't add the friend again if they are already in the list!
- Note that you actually get a reference to the friend list, so you can add the friend directly to the list and don't have to re-populate the original database (line 15)

Lecture 6: Lab Review

- The **`unfriend()`** function:

```python
def unfriend(friend_database, dwarf, current_friend):
    """
    Unfriends current_friend from dwarf's friend list.
    :param friend_database: The database of friends
    :param dwarf: The dwarf whose friend will be unfriended
    :param current_friend: The friend of the dwarf who will be unfriended
    :return: True if the dwarf had the friend, False if the dwarf did not
    have the friend
    """
    pass
```

Let's use **`try/except`**! How does a list removal fail? I don't know -- let's check:

```
>>> a = [1,5,7]
>>> a.remove(5)
>>> a.remove(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

It looks like we have a **`ValueError`**.

Lecture 6: Lab Review

- The **`unfriend()`** function:

```python
def unfriend(friend_database, dwarf, current_friend):
    """
    Unfriends current_friend from dwarf's friend list.
    :param friend_database: The database of friends
    :param dwarf: The dwarf whose friend will be unfriended
    :param current_friend: The friend of the dwarf who will be unfriended
    :return: True if the dwarf had the friend, False if the dwarf did not
    have the friend
    """
    friend_list = friend_database[dwarf] # get friend list from database
    try:
        friend_list.remove(current_friend)
        return True
    except ValueError:
        print(f"{current_friend} is not your friend! No change made.")
        return False
```

- With **`try/except`** it is a very straightforward function.

Lecture 6: Lab Review

- The **`save_friends()`** function:

```python
def save_friends(friend_database, filename):
    """
    Saves friend_database to filename
    :param friend_database: The friend database
    :param filename: The file to save the database to
    :return: True if the database was saved, False otherwise
    """
    pass
```

- We will overwrite the filename

Lecture 6: Lab Review

- The **`save_friends()`** function:

```python
def save_friends(friend_database, filename):
    """
    Saves friend_database to filename
    :param friend_database: The friend database
    :param filename: The file to save the database to
    :return: True if the database was saved, False otherwise
    """
    try:
        with open(filename, "w") as f:
            for dwarf, friends in friend_database.items():
                f.write(" ".join([dwarf] + friends) + '\n')
        return True
    except FileNotFoundError:
        return False
```

- This is the "one-liner" write statement, but you have to understand a few things:
  - Adding two lists together concatenates them (and we have to create a list out of the dwarf by surrounding it with brackets)
  - The join function with a space puts the space between all the values
  - Adding two strings concatenates the strings
- That's a lot to do in one line! Let's make it a bit more verbose (but okay)

Lecture 6: Lab Review

- The **`save_friends()`** function:

```python
def save_friends(friend_database, filename):
    """
    Saves friend_database to filename
    :param friend_database: The friend database
    :param filename: The file to save the database to
    :return: True if the database was saved, False otherwise
    """
    try:
        with open(filename, "w") as f:
            for dwarf, friends in friend_database.items():
                f.write(dwarf)
                f.write(" ")
                for friend in friends[:-1]:  # handle last as special case (no space)
                    f.write(friend)
                    f.write(" ")
                f.write(friends[-1])  # last does not have a space
                f.write("\n")  # but it does have a newline
        return True
    except FileNotFoundError:
        return False
```

- This is more verbose, but easier to understand if you go one line at a time

Lecture 6: Course Recap - what have we covered?

- Here are the topics that we have covered so far, and that will be on the midterm next Tuesday. There has been a lot to learn!
    - Week 1:
        - Arithmetic, Variables, The Python REPL
    - Week 2:
        - Functions, Doctests, Passing values to functions, Branching
    - Week 3:
        - Iteration, Lists
    - Week 4:
        - Tuples, List slicing, List comprehensions, Strings
    - Week 5:
        - Recursion and Dictionaries
    - Week 6:
        - File processing and exception handling

Lecture 6: Course Recap - what have we covered?

- Week 1: Arithmetic, Variables, The Python REPL
  - Sample arithmetic midterm question:
    - Write a function that has three parameters **`a`**, **`b`**, and **`c`**, and returns **`a`** multiplied by **`b`** and then added to **`c`**:

```python
def linear_combination(a, b, c):
  """
  :return The value of a multiplied by b and added to c
  """
  pass
```

Lecture 6: Course Recap - what have we covered?

- Week 1: Arithmetic, Variables, The Python REPL
  - Sample arithmetic midterm question:
    - Write a function that has three parameters **`a`**, **`b`**, and **`c`**, and returns **`a`** multiplied by **`b`** and then added to **`c`**:

```python
def linear_combination(a, b, c):
  """
  :return The value of a multiplied by b and added to c
  """
  return a * b + c
```

Lecture 6: Course Recap - what have we covered?

- Week 1: Arithmetic, Variables, The Python REPL
  - Sample variables and REPL question:
    - Multiple choice: What is the next line in the REPL going to print?

```
>>> def first_and_last(lst):
...   first = lst[0]
...   last = lst[-1]
...   return (first, last)
...
>>> result = first_and_last([5,10,15,20])
>>> print(result)
(5, 20)
>>> print(first)
```

A.
```
5
>>>
```
B.
```
10
>>>
```
C.
```
[5, 10, 15, 20]
>>>
```
D.
```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'first' is not defined
>>>
```

Lecture 6: Course Recap - what have we covered?

- Week 1: Arithmetic, Variables, The Python REPL
  - Sample variables and REPL question:
    - Multiple choice: What is the next line in the REPL going to print?

```
>>> def first_and_last(lst):
...   first = lst[0]
...   last = lst[-1]
...   return (first, last)
...
>>> result = first_and_last([5, 10, 15, 20])
>>> print(result)
(5, 20)
>>> print(first)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'first' is not defined
>>>
```

- The answer was (D). This is something we probably didn't go over enough -- variables inside a function are *local*, and are not available outside the function (this is called *scoping*).
- In other words: you cannot access a variable defined in another function. This is a good thing! It means that you can use as many variables inside a function, and when you use the function, you don't have to know those names.

Lecture 6: Course Recap - what have we covered?

- Week 2: Functions, Doctests, Passing values to functions, Branching

- Lots of potential functions questions.
  - You should be able to write a doctest for any function that does not have user input, open a file, etc.
  - Sample functions question (this would be a challenging problem)

A. What does **`mystery_func`** do?

B. Write a non-trivial doctest for **`mystery_func`**.

C. What does **`main()`** print out for this program?

```python
def mystery_func(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

def main():
    count = 0
    lst = [2, 3, 4]
    for num in lst:
        if mystery_func(num):
            count += 1

    if count == 0:
        print("None")
    elif count < len(lst):
        print("Some")
    else:
        print("All")


if __name__ == "__main__":
    main()
```

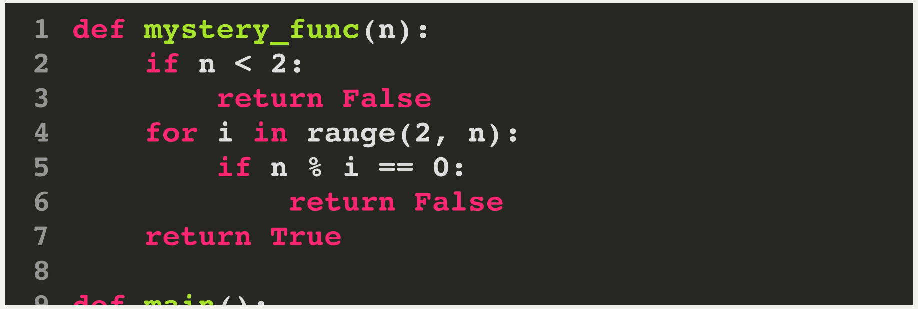
Lecture 6: Course Recap - what have we covered?

- Week 2: Functions, Doctests, Passing values to functions, Branching

A. What does **`mystery_func`** do?

B. Write two non-trivial doctests for **`mystery_func`**.

C. What does **`main()`** print out for this program?



```python
1 def mystery_func(n):
2     if n < 2:
3         return False
4     for i in range(2, n):
5         if n % i == 0:
6             return False
7     return True
8
9 def main():
```

To solve this problem, I suggest trying a few numbers in the function, and going through each line. Let's try **`mystery_func(5)`**:

- **`n`** is **`5`**, and is not less than **`2`**, so skip line 3
- On line 4, the range is **`2,3,4`**
- On line 5, when **`i==2`**, **`5 % 2`** is **`1`**, so we continue in the loop:
  - when **`i==3`**, **`5 % 3`** is **`1`**, so we continue in the loop
  - when **`i==4`**, **`5 % 4`** is **`1`**, so we finish the loop
- The loop is complete, so we return **`True`**

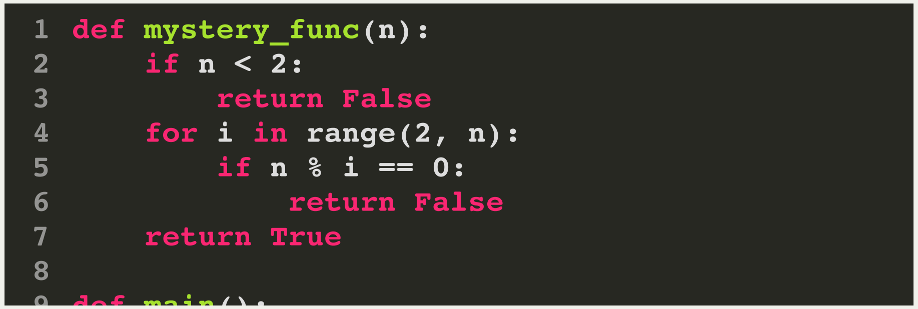
Lecture 6: Course Recap - what have we covered?

- Week 2: Functions, Doctests, Passing values to functions, Branching

A. What does **`mystery_func`** do?

B. Write two non-trivial doctests for **`mystery_func`**.

C. What does **`main()`** print out for this program?



```
1 def mystery_func(n):
2     if n < 2:
3         return False
4     for i in range(2, n):
5         if n % i == 0:
6             return False
7     return True
8
9 def main():
```

Let's try **`mystery_func(9)`**:

- **`n`** is **`9`**, and is not less than **`2`**, so skip line 3
- On line 4, the range is **`2,3,4,5,6,7,8`**
- On line 5, when **`i==2`**, **`9 % 2`** is **`1`**, so we continue in the loop:
  - when **`i==2`**, **`9 % 3`** is **`0`**, so we return **`False`**

Based on **`mystery_func(3)`** returning **`True`** and **`mystery_func(9)`** returning **`False`**, can you figure out what the function is doing? If not, try some more:

`mystery_func(2)` returns **`False`**, **`mystery_func(3)`** returns **`True`**, **`mystery_func(4)`** returns **`False`**, **`mystery_func(6)`** returns **`False`**, **`mystery_func(7)`** returns **`True`**.

Lecture 6: Course Recap - what have we covered?

- Week 2: Functions, Doctests, Passing values to functions, Branching

A. What does **`mystery_func`** do?

B. Write two non-trivial doctests for **`mystery_func`**.

C. What does **`main()`** print out for this program?

```python
def mystery_func(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

```

At this point, hopefully you can see that the function returns whether or not **`n`** is prime.

The doctests:

```python
def mystery_func(n):
    """
    >>> mystery_func(7)
    True
    >>> mystery_func(6)
    False
    """
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

```python
def main():
    count = 0
    lst = [2, 3, 4]
    for num in lst:
        if mystery_func(num):
            count += 1

    if count == 0:
        print("None")
    elif count < len(lst):
        print("Some")
    else:
        print("All")


```

`main()` prints:

`Some`

(because **2** and **4** are not prime, and **3** is prime.

Lecture 6: Course Recap - what have we covered?

- Week 2: Functions, Doctests, Passing values to functions, Branching

- Potential branching question:
  - Write the output for the following values of the parameters `a`, `b`, and `c`. (2 pts. each)
    - i. a: 3, b: 3, c: 9
    - ii. a: 8, b: 7, c: 9
    - iii. a: 1, b: 4, c: 4
    - iv. a: 4, b: 7, c: 2
    - v. a: 2, b: 4, c: 7

```python
def batman(a, b, c):
    if a > 3:
        if a < c:
            print("Robin")
        else:
            print("Batman")
    elif b == 4:
        if b != c:
            print("Batmobile")
        else:
            print("Toolbelt")
    else:
        print("Alfred")
```

Lecture 6: Course Recap - what have we covered?

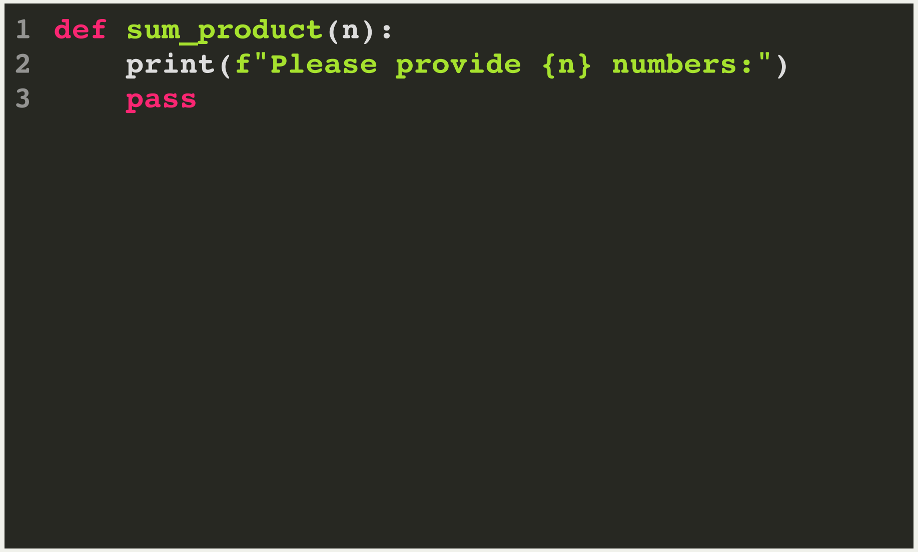
- Week 2: Functions, Doctests, Passing values to functions, Branching

- Potential branching question:
  - Write the output for the following values of the parameters `a`, `b`, and `c`. (2 pts. each)
    - i. a: 3, b: 3, c: 9 **Alfred**
    - ii. a: 8, b: 7, c: 9 **Robin**
    - iii. a: 1, b: 4, c: 4 **Toolbelt**
    - iv. a: 4, b: 7, c: 2 **Batman**
    - v. a: 2, b: 4, c: 7 **Batmobile**

```python
def batman(a, b, c):
    if a > 3:
        if a < c:
            print("Robin")
        else:
            print("Batman")
    elif b == 4:
        if b != c:
            print("Batmobile")
        else:
            print("Toolbelt")
    else:
        print("Alfred")
```

Lecture 6: Course Recap - what have we covered?

- Week 3: Iteration, Lists
  - You should be able to iterate through a list, and you should be able to iterate with a range. You should be able to write **`while`** and **`for`** loops for your code. You should also understand how the **`enumerate`** function works.
  - Sample iteration problem:
    - Write a function that asks a user for n numbers and then returns the sum and product of the numbers as a Tuple:



```python
def sum_product(n):
    print(f"Please provide {n} numbers:")
    pass
```

- Sample run:

```
Please provide 4 numbers:
Number 0: 2
Number 1: 4
Number 2: 5
Number 3: 7
Sum: 18, Product: 280
```

Lecture 6: Course Recap - what have we covered?

- Week 3: Iteration, Lists
  - You should be able to iterate through a list, and you should be able to iterate with a range. You should be able to write **`while`** and **`for`** loops for your code. You should also understand how the **`enumerate`** function works.
  - Sample iteration problem:
    - Write a function that asks a user for n numbers and then returns the sum and product of the numbers as a Tuple:

```python
def sum_product(n):
    print(f"Please provide {n} numbers:")
    sum = 0
    product = 1
    for i in range(n):
        next_num = int(input(f"Number {i}: "))
        sum += next_num
        product *= next_num
    return sum, product
```

- Sample run:

```
Please provide 4 numbers:
Number 0: 2
Number 1: 4
Number 2: 5
Number 3: 7
Sum: 18, Product: 280
```

Lecture 6: Course Recap - what have we covered?

- Week 3: Iteration, Lists
    - Sample lists problem:
        - Write a function that counts the number of elements in a list within a specified range:

```
def count_list_range(lst, min, max):
   """
   :return a count of numbers in the list
   between (and including) min and max
   """
```

- Example:

```
>>> print(count_list_range([1, 5, 2, 11, 6, 18, 4, 9], 5, 10))
3
```

Lecture 6: Course Recap - what have we covered?

- Week 3: Iteration, Lists
  - Sample lists problem:
    - Write a function that counts the number of elements in a list within a specified range:

```python
def count_list_range(lst, min, max):
    """
    :return a count of numbers in the list
    between (and including) min and max
    """
    count = 0
    for value in lst:
        if min <= value <= max:
            count += 1
    return count
```

- One-liner solution:

```python
def count_list_range(lst, min, max):
    """
    :return a count of numbers in the list
    between (and including) min and max
    """
    return len([x for x in lst if min <= x <= max])
```

Lecture 6: Course Recap - what have we covered?

- Week 3: Iteration, Lists
  - Sample lists problem 2:
    - Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division (e.g., **`a // b`**) to produce the final average. You may assume that the array is length 3 or more. You can use the **`sum()`**, **`min()`**, and **`max()`** functions.

```python
def centered_average(lst):
  pass
```

Lecture 6: Course Recap - what have we covered?

- Week 3: Iteration, Lists
    - Sample lists problem 2:
        - Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division (e.g., **`a // b`**) to produce the final average. You may assume that the array is length 3 or more. You can use the **`sum()`**, **`min()`**, and **`max()`** functions.

```python
def centered_average(lst):
    pass
```

Lecture 6: Course Recap - what have we covered?

- Week 3: Iteration, Lists
    - Sample lists problem 2:
        - Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division (e.g., **`a // b`**) to produce the final average. You may assume that the array is length 3 or more. You can use the **`sum()`**, **`min()`**, and **`max()`** functions.

```python
def centered_average(lst):
    lst_cpy = list(lst)
    lst_cpy.remove(min(lst_cpy))
    lst_cpy.remove(max(lst_cpy))
    return sum(lst_cpy) // len(lst_cpy)
```

Lecture 6: Course Recap - what have we covered?

- Week 4: Tuples, List slicing, List comprehensions, Strings
  - Remember, tuples cannot be modified. Often, we return a tuple as a return value, to return more than one value (as tuple elements)
  - List comprehensions always return another list
  - Strings are iterable, and each character has an ASCII value
  - Sample list comprehension problem:
    - Use a list comprehension to generate a list of all the numbers between 1 and n that are divisible by 7

```python
def div_by_seven(n):
  pass
```

Lecture 6: Course Recap - what have we covered?

- Week 4: Tuples, List slicing, List comprehensions, Strings
  - Remember, tuples cannot be modified. Often, we return a tuple as a return value, to return more than one value (as tuple elements)
  - List comprehensions always return another list
  - Strings are iterable, and each character has an ASCII value
  - Sample list comprehension problem:
    - Use a list comprehension to generate a list of all the numbers between 1 and n that are divisible by 7

```python
def div_by_seven(n):
    return [x for x in range(1, n) if x % 7 == 0]
```

Lecture 6: Course Recap - what have we covered?

- Week 4: Tuples, List slicing, List comprehensions, Strings
  - Sample list comprehension problem 2:
    - Use a list comprehension to remove all the vowels in a string (don't forget to join the string back together once you have the list!)

```python
def remove_vowels(str):
    pass
```

Lecture 6: Course Recap - what have we covered?

- Week 4: Tuples, List slicing, List comprehensions, Strings
    - Sample list comprehension problem 2:
        - Use a list comprehension to remove all the vowels in a string (don't forget to join the string back together once you have the list!)

```python
def remove_vowels(str):
    return "".join([x for x in str if x.lower() not in "aeiou"])
```

Lecture 6: Course Recap - what have we covered?

- Week 4: Tuples, List slicing, List comprehensions, Strings
    - Sample list comprehension problem 3:

```python
def words_not_the(sentence):
    """Words not 'the'
    Given a sentence, produce a list of the lengths of each word in the sentence,
    but only if the word is not 'the'.
    >>> words_not_the('the quick brown fox jumps over the lazy dog')
    [5, 5, 3, 5, 4, 4, 3]
    """
    pass
```

Lecture 6: Course Recap - what have we covered?

- Week 4: Tuples, List slicing, List comprehensions, Strings
  - Sample list comprehension problem 3:

```python
def words_not_the(sentence):
    """Words not 'the'
    Given a sentence, produce a list of the lengths of each word in the sentence,
    but only if the word is not 'the'.
    >>> words_not_the('the quick brown fox jumps over the lazy dog')
    [5, 5, 3, 5, 4, 4, 3]
    """
    return [len(x) for x in sentence.split(' ') if x != 'the']
```

Lecture 6: Course Recap - what have we covered?

- Week 5: Recursion and Dictionaries
  - You should be able to trace rudimentary recursive functions, and write simple recursive functions
  - You should understand how to create dictionaries, and how to access the key/value pairs
  - Example recursion problem:
    - Write a recursive function that returns the sum of the ints from 1 to n

```python
def recursive_sum(n):
  pass
```

Lecture 6: Course Recap - what have we covered?

- Week 5: Recursion and Dictionaries
  - You should be able to trace rudimentary recursive functions, and write simple recursive functions
  - You should understand how to create dictionaries, and how to access the key/value pairs
  - Example recursion problem:
    - Write a recursive function that returns the sum of the ints from 1 to n

```python
def recursive_sum(n):
    if n == 1:
        return 1
    return n + recursive_sum(n - 1)
```

Lecture 6: Course Recap - what have we covered?

- Week 5: Recursion and Dictionaries
  - You should be able to trace rudimentary recursive functions, and write simple recursive functions
  - You should understand how to create dictionaries, and how to access the key/value pairs
  - Example recursion problem 2:
    - What does the following recursive function do?

```python
def mystery(i):
    if i < 1:
        return
    print(i)
    printFun(i - 1)
    print(i)
```

Lecture 6: Course Recap - what have we covered?

- Week 5: Recursion and Dictionaries
    - You should be able to trace rudimentary recursive functions, and write simple recursive functions
    - You should understand how to create dictionaries, and how to access the key/value pairs
    - Example recursion problem 2:
        - What does the following recursive function do?

```python
def mystery(i):
    if i < 1:
        return
    print(i)
    printFun(i - 1)
    print(i)
```

The function prints `i` down to `1`, and then `1` back up to `i`.

Example:

```
printFun(3)
3
2
1
1
2
3
```

Lecture 6: Course Recap - what have we covered?

- Week 5: Recursion and Dictionaries
  - Example dictionary problem:
    - Given a list of integers, produce a dictionary that counts the number of values between 0 and 9, 10 and 19, and 20 and 29. The keys should be 0, 10, and 20, and the values should be the counts in the ranges above.

Example:

```
>>> histogram([12, 5, 3, 19, 28, 15, 14, 1, 23])
{0: 3, 10: 4, 20: 2}
```

```python
def histogram(lst):
    pass
```

Lecture 6: Course Recap - what have we covered?

- Week 5: Recursion and Dictionaries
  - Example dictionary problem:
    - Given a list of integers, produce a dictionary that counts the number of values between 0 and 9, 10 and 19, and 20 and 29. The keys should be 0, 10, and 20, and the values should be the counts in the ranges above.

Example:

```
>>> histogram([12, 5, 3, 19, 28, 15, 14, 1, 23])
{0: 3, 10: 4, 20: 2}
```

Short, generic solution:

```python
def histogram(lst):
    hist = {}
    for value in lst:
        if value // 10 * 10 not in hist:
            hist[value // 10 * 10] = 1
        else:
            hist[value // 10 * 10] += 1
    return hist
```

Verbose solution:

```python
def histogram2(lst):
    hist = {}
    for value in lst:
        if 0 <= value < 10:
            if 0 not in hist:
                hist[0] = 1
            else:
                hist[0] += 1
        elif 10 <= value < 20:
            if 10 not in hist:
                hist[10] = 1
            else:
                hist[10] += 1
        elif 20 <= value <= 30:
            if 20 not in hist:
                hist[20] = 1
            else:
                hist[20] += 1
    return hist
```

Lecture 6: Course Recap - what have we covered?

- Week 6: File processing and exception handling
  - You should be able to open files to read or write, and you should be able to read from them and write to them. You should be using one of the following patterns:

```python
with open(filename, "r") as f:
  for line in f:
    line = line[:-1]  # remove newline
    # do something with the line
```

```python
with open(filename, "r") as f:
  for line in f:
    allText = f.readlines()  # read all lines into a list (will still have newlines)
    allText = [x[:-1] for x in allText]  # remove newlines (if necessary)
    # now allText has a list of all the lines in the file with no newlines at the end
```

```python
with open(filename, "r") as f:
  for line in f:
    allText = f.read() # read all data
    # now allText has the entire file
```

```python
with open(filename, "w") as f:
    f.write(mydata)
    f.write('\n')  # each line should end with a newline
```

Lecture 6: Course Recap - what have we covered?

- Week 6: File processing and exception handling
  - Example file processing problem:
    - Reverse the lines in a file. Do not provide error checking.

```python
def reverse_lines(filename):
        pass
```

Lecture 6: Course Recap - what have we covered?

- Week 6: File processing and exception handling
  - Example file processing problem:
    - Reverse the lines in a file. Do not provide error checking.

```python
def reverse_lines(filename):
        with open(filename, "r") as f:
        lines = f.readlines()

    lines = lines[::-1]
    with open(filename, "w") as f:
        for line in lines:
            f.write(line)
```

Lecture 6: Course Recap - what have we covered?

- Week 6: File processing and exception handling
  - For exception handling, you should be able to use **`try/except`** to catch errors. We will give you the error types (e.g., **`FileNotFoundError`**)
  - Example exception handling problem:
    - Write a function that returns the value for a given key in a dictionary, and returns None if the value does not exist.

```python
def get(dict, key):
    pass
```

Lecture 6: Course Recap - what have we covered?

- Week 6: File processing and exception handling
  - For exception handling, you should be able to use **`try/except`** to catch errors. We will give you the error types (e.g., **`FileNotFoundError`**)
  - Example exception handling problem:
    - Write a function that returns the value for a given key in a dictionary, and returns None if the value does not exist.

```python
def get(dict, key):
    try:
      return dict[key]
    except KeyError:
      return None
```

- It turns out that this function already exists in the dictionary definition!

```
help(dict.get)

get(self, key, default=None, /)
    Return the value for key if key is in the dictionary, else default.
```

- You use it like this:

```
>>> d = {'name': 'Chris'}
>>> age = d.get('age')
>>> print(age)
None
>>> age = d['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
>>>
```

Lecture 6: Course Recap - what have we covered?

- We could have made our original letter histogram easier. Original:

```python
def letter_histogram(s):
  freqs = {}
  for c in s:
    if c not in freqs:
      freqs[c] = 1
    else:
      freqs[c] += 1

   return freqs
```

- Version with `get()`:

```python
def letter_histogram(s):
  freqs = {}
  for c in s:
    freqs[c] = freqs.get(c, 0) + 1

   return freqs
```

```
>>> letter_histogram("the quick brown fox jumps over the lazy dog")
{'t': 2, 'h': 2, 'e': 3, ' ': 8, 'q': 1, 'u': 2, 'i': 1, 'c': 1, 'k': 1, 'b': 1, 'r': 2, 'o': 4,
 'w': 1, 'n': 1, 'f': 1, 'x': 1, 'j': 1, 'm': 1, 'p': 1, 's': 1, 'v': 1, 'l': 1, 'a': 1, 'z': 1,
 'y': 1, 'd': 1, 'g': 1}
```

Lecture 6: Studying for the Midterm

1. Check out the Midterm study guide: https://tinyurl.com/cs5001-midterm
2. Review all lecture slides and examples, assignments, and labs
3. Practice! Go through the practice problems we went over in class today, and the ones you will see in lab. You should be able to answer every practice problem correctly!
4. Look at the reference sheet that we will give you during the exam: https://tinyurl.com/cs5001-midterm-reference
5. Practice some more!

The midterm will be 2.5 hours long, from 7pm-9:30pm next Tuesday. It will be using a computerized testing system called BlueBook -- more details about that will follow.