Lab 7

- Lab 6 Review
- Review for Lab 7

Lab Musings

- As we learn more computer science, we're moving toward a much **higher ratio of thinking to coding**
  - Give yourself the time and room to think
- Going beyond simply correctness in solutions
  - Looking for understanding of good coding practices
    - Testing, readability, usability, documentation, organization, efficiency
      - (not necessarily in that order)

Lab Musings

- Lab benefit: access to other students, lab assistants, and instructor to help
- Lab limitation: may not be the best environment
  - Seems to cause a competitive atmosphere, increased anxiety for some students
  - You have until Friday to complete the lab
  - Work at your pace, **think clearly** and **deeply**

Compare Solutions

```
words = sentence.split()

shorthandList = []
for word in words:
    shorthandList.append(word[0])

shorthand = "".join(shorthandList)

shorthand = shorthand.lower()

print("Shorthand is:", shorthand)
```

```
words = sentence.split()

shorthand=""
for word in words:
    shorthand += word[0]

shorthand = shorthand.lower()

print("Shorthand is:", shorthand)
```

Compare Solutions

```
words = sentence.split()

shorthandList = []
for word in words:
    shorthandList.append(word[0])

shorthand = "".join(shorthandList)

shorthand = shorthand.lower()

print("Shorthand is:", shorthand)
```

Both are valid solutions. I'm not sure which is more efficient in practice.

However, the solution at left has more conceptual complexity (appending to a list and then converting to a string, as opposed to just creating the string).

In general, looking for less complex solutions.

Saw similar, more complex solutions for the password generation problem.

```
words = sentence.split()

shorthand=""
for word in words:
    shorthand += word[0]

shorthand = shorthand.lower()

print("Shorthand is:", shorthand)
```


Generating a Random Password

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

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

for charPos 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)

Consider:

```
MIN_NUM=ord('0')
```


Review Caesar Cipher

- Consider the following solutions

```
for char in message:
    asciiVal = ord(char)
    if asciiVal == 32:
        …
    else:
        …
```

Which is easier to read and understand?

```
for char in message:
    if char == " ":
        …
    else:
        …
```


Review Caesar Cipher

- Consider the following solutions

```
for char in message:
    asciiVal = ord(char)
    if asciiVal == 32:
        …
    else:
        …
```

I know what " " means. I don't immediately know what 32 means.

Lesson: prefer words over numbers.

```
for char in message:
    if char == " ":
        …
    else:
        …
```

Caesar Cipher with Files

- High-level description explaining what you're doing at the top of the program
- How to debug
  - Look at the input files
- Common issues
  - Not handling new lines ("\n") in the file
    - Similar to handling spaces
  - Close files as soon as possible

Review

- What is the keyword we use to create a new function?
- How do we get output from a function?
- What happens in the program execution when a function reaches a `return` statement?
- Why do we write functions?
- Why do we write functions?
- What makes a good function?
- How should you comment your functions?
- What is the name for the process for changing a program to improve readability/organization/readability without changing functionality?

Review: Writing a "Good" Function

- Should be an "intuitive chunk"
  - Doesn't do too much or too little
  - If does too much, try to break into more functions
- Should be reusable
- Always have comment that tells what the function does

Writing Comments for Functions

- Good style: Each function ***must*** have a comment
  - Describes functionality at a high-level
  - Include the *precondition*, *postcondition*
  - Describe the parameters (their types) and the result of calling the function (precondition and postcondition may cover this)

Writing Comments for Functions

- Include the function's pre- and post- conditions
- **Precondition**: Things that must be true for function to work correctly
  - E.g., num must be even
- **Postcondition**: Things that will be true when function finishes (if precondition is true)
  - E.g., the returned value is the max

Example Comment

- Describes at high-level
- Describes parameters

```
def printVerse(animal, sound):
    """
    Prints a verse of Old MacDonald, plugging in the
    animal and sound parameters (which are strings),
    as appropriate.
    """
    print(BEGIN_END + EIEIO)
    print("And on that farm he had a " + animal + EIEIO)
    …
```

Comment style: **Docstring** "documentation string"

Comments from docstrings show up when you use `help` function

Pre/Post Conditions

```
def binaryToDecimal( binary_string ):
    """
    pre: binary_string is a string that contains
    only 0s and 1s
    post: returns the decimal value for the binary
    string
    """
    dec_value = 0
    for pos in range( len( binNum ) ):
        exp = len(binNum) - pos - 1
        bit = int(binNum[pos])

        # compute the decimal value of this bit
        val = bit * 2 ** exp

        # add it to the decimal value
        decVal += val

    return dec_value
```


Function comments

```
def printHeadings():
    """displays table column headings"""
```

Good. Describes function at high level

```
def printHeadings():
    """defines the printHeader function"""
```

Not descriptive.
Says what *you're* doing, not what ***function*** does
Need to tell programmer how to use function

Summary "Good" Function

- Reusable functionality
- Good function name
- Good parameter names
- Good documentation
  - Well-described input, output

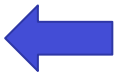
Review: Refactoring

Converting Functionality into Functions

1. Identify functionality that should be put into a function
   - What is the function's input?
   - What is the function's output?
2. Define the function
   - Write comments
3. Call the function where appropriate
4. Create a `main` function that contains the "driver" for your program
   - Put at top of program
5. Call `main` at bottom of program

TOP-DOWN DESIGN

Designing Code

- 1st Approach: Bottom-up
  - Create functions
  - Call functions
- 2nd Approach: Refactoring
  - Write code
  - Refactor code to have functions
  - Call those functions
- 3rd approach: Top-down Design 
  - Write code, calling functions
  - Write "stub" functions
  - Fill-in functions later

Top-Down Design:
Alternative Approach to Development

1. Create overview, e.g., in `main`
2. Define functions later

```
def main():
    # get the binary number from the user, as a string
    binNum = input("Please enter a binary number: ")
    isBinary = checkBinary(binNum)
    if not isBinary : # equivalent to isBinary == False
        print(binNum, "is not a binary number.")
        sys.exit()

    decVal = binaryToDecimal(binNum)
    print(binNum, "is", decVal)
```

Benefits:
- Know what functions you need
- Know the requirements for your functions
  - What is each function's input, output

Problem: Create a Summary Report

- **Given**: a file containing students names and their years (first years, sophomore, junior, or senior) for this class
- **Problem**: create a report (in a file) that says the year and how many students from that year are in this class, on the same line.

`writeSumReport.py`

Development Advice

- Build up your program in steps
  - Always write small pieces of code
  - Test *function* separately from other code, using a test function
  - Test, debug. **Repeat**
- Development Options:
  - Refactor:
    - Write function body as part of **`main`**, test
    - Then, separate out into its own function
  - Top-down design
  - Bottom-up design

May use more than one approach in a program

Example: Could still refactor after using these options

Lab 7

- Function practice
- Defining functions (refactoring)
- File practice
- Working with lists

Testing Functions

1. Create test cases
   - Input, expected output
2. Write a function that creates lists of the input and expected output and automatically tests your function
3. Call the function to test your function
4. Iterate
   - Add additional test cases if needed to help debug your function

Review: Testing Functions

```
def testBinaryToDecimal():
    """Test the binaryToDecimal function.
       Displays the correctness or incorrectness of the
       function.
       Nothing is returned."""

    paramInputs = ["0", "1", "10", "1001", "10000"]
    expectedResults = [ 0, 1, 2, 9, 16]
    for index in range(len(paramInputs)):
        paramInput = paramInputs[index]
        expectedResult = expectedResults[index]
        actualResult = binaryToDecimal(paramInput)
        if actualResult != expectedResult:
            print("**ERROR!**", paramInput, "should be", \
                  expectedResult)
            print("Instead, got", actualResult)
        else:
            print("Success on binary to decimal conversion for",\
                  paramInput, "-->", actualResult)
```

Call function to test: `testBinaryToDecimal()`

Getting Documentation

- **`dir`**: function that returns a list of methods and attributes in an object
  - `dir(<type>)`
- **`help`**: get documentation

- In the Python shell
  - `help(<type>)`
  - `import <modulename>`
  - `help(<modulename>)`

Problem: Create a Summary Report

- **Given**: a file containing students names and their years (first years, sophomore, junior, or senior) for this class
- **Problem**: create a report (in a file) that says the year and how many students from that year are in this class, on the same line.

Pseudocode for program

```
def main():
    # get name of data file
    # open output file
    for searchTerm in searchTerms:
        numFound = numOccurrences( searchTerm, dataFileName )
        outputFile.write("%s %d\n" % (searchTerm, numFound))
    # close output file
```

Example of top-down design:
- Can fill in details, e.g., the comments, the function numOccurrences

Gymnastics Scores

- Read in first line of file
  - Can use `readline()` method
- Read in rest of lines
  - Either a `for` or `while` loop

Gymnast Scores (Partial Solution)

```
judgeFile = file(FILENAME, "r")

avgDifficulty = judgeFile.readline()
avgDifficulty = float(avgDifficulty)

min = 10
max = 0
total = 0

for x in xrange(6):  # get next 6 execution scores
    line = judgeFile.readline()
    score = float(line)
    if score < min:
        min = score
    if score > max:
        max = score
    total += score
judgeFile.close()
total -= max + min  # exclude high and low scores from total
…
```

Read in separately, Not in loop → inefficient

Keep track of "current" min/max

Comments: what code *means*

Review

- What does x represent and what is its data type for the following code snippets?

```
y = "computers"
z = [1, 2, 5, 7]

for x in y:

for x in range(len(y)):

for x in z:

for x in range(len(z)):
```


DEAL OR NO DEAL

Lab 7: *Deal or No Deal* Overview

- Have 26 cases with various amounts of money
  - Amounts are known
- Player selects a case (hope has the big jackpot)
- In each round, player opens up cases
  - Reveals amounts that are not in the case they chose
- Banker makes an offer to buy the case
- Player decides if want to take the deal
  - Is the offer more than what is in the case?
  - Make decision based on amounts that haven't been opened yet
- Game ends when only one more case to open (two amounts on board) or player takes the deal.

Implementing *Deal or No Deal*

- Given: partial solution in code
  - `main()` function, some additional functions are already written
- Your job:
  - Read, understand given code
  - Fill in the functions for a complete solution
- Example of top-down design
  - In main() …

`printBoard` not yet defined

```
# keep track of how much was in your case
# and mark the case as chosen.
amtInCase = cases[choice]
cases[choice] = CHOSEN
printBoard(caseValues)
```


Modeling *Deal or No Deal*

How can we represent that a case has been opened?

- Cases, numbered 0 to 25
  - Have dollar amounts in them

1000000	1000	5	…	750000	value
0	1	2	…	25	case/ position

- Board
  - Which dollar amounts have been chosen, which are still in play

.01	1	5	…	1000000	value
0	1	2	…	25	position

Modeling *Deal or No Dea*

CHOSEN = -1
means case opened:
Don't display on board,
Don't allow user to select again

- Cases, numbered 0 to 25
  - Have dollar amounts in them

1000000	1000	5	…	CHOSEN	value
0	1	2	…	25	case/ position

- Board
  - Which dollar amounts have been chosen, which are still in play

.01	CHOSEN	5	…	1000000	value
0	1	2	…	25	position

Functionality

- Read in values contained in cases from a file
  - What data type should these values be?
- Have user select from remaining cases
  - Make sure choice is valid
- Display remaining cases
  - Print four to a row
- Display remaining amounts on board
  - Left column is smaller amounts

Advice: don't worry about the formatting at first
Do a first pass implementation of all the functions, then go back and refine the functions.

How to print remaining cases?

- Cases, numbered 0 to 25
  - Have dollar amounts in them

1000000	1000	5	…	CHOSEN	value
0	1	2		25	case/ position

- Board
  - Which dollar amounts have been chosen, which are still in play

.01	CHOSEN	1000	…	-1	value
0	1	2		25	position

Honor System Review

- Person who needs help should *never* look at the code of the person who is helping
- No sharing code
  - No emailing, printing, …
- Cite the help you're receiving outside of lab
- Pledge your assignments
- Report suspicious behavior

Rules for Collaboration

- Debugging help
  - 5 minute rule: a friend can only look at your code *to help with debugging* for 5 minutes
  - Owner of code owns keyboard/mouse
- Problem solving discussion
  - No written solutions leave the room
- Acknowledge aid
- Do not give out your password

Lab 7 Overview

- Focus: program organization
  - Defining and Using Functions
- Deal or No Deal

Debugging

Error Message:
NameError: global name 'num' is not defined

```
def binaryToDecimal(binary):
    # accumulate the decimal value in this variable
    decVal = 0

    # go through the positions in the string
    for pos in xrange(len(num)):
        # num[pos] is a string; need to convert to an int
        bit = int(num[pos])
        # calculate which "place" the current bit is at
        place = 2**(len(num)-pos-1)
        # add to the decimal value
        decVal += place * bit
    return decVal
```