CMSC201
Computer Science I for Majors

Lecture 18 – Program Design (Continued)

Last Class We Covered

- File I/O
  - How to open a file
    - For reading or writing
  - How to write to a file
  - How to close a file
- The `join()` function

Any Questions from Last Time?

Today's Objectives

- To discuss the details of "good code"
  - Readability
  - Adaptability
- To learn the "rules" of commenting
- To learn how to design a program
- To learn more about Incremental Programming

Motivation

- We've talked a lot about certain 'good habits' we'd like you all to get in while writing code
  - What are some of them?

- There are two main reasons for these habits
  - Readability
  - Adaptability

“Good Code” – Readability

Readability

- Having your code be readable is important, both for your sanity and anyone else's
  - Your TA's sanity is important

- Having highly readable code makes it easier to:
  - Figure out what you're doing while writing the code
  - Figure out what the code is doing when you come back to look at it a year later
  - Have other people read and understand your code

Improving Readability

- Improving readability of your code can be accomplished in a number of ways
  - Comments
  - Meaningful variable names
  - Breaking code down into functions
  - Following consistent naming conventions
  - Programming language choice
  - File organization

Readability Example

- What does the following code snippet do?

```
def nS(p, c):
    l = len(p)
    if (l >= 4):
        c += 1
        print(p)
        if (l >= 9):
            return p
    # FUNCTION CONTINUES...
```

- There isn't much information to go on, is there?

Readability Example

- What if I used meaningful variable names?

```
def nS(p, c):
    l = len(p)
    if (l >= 4):
        c += 1
        print(p)
        if (l >= 9):
            return p
    # FUNCTION CONTINUES...
```

Readability Example

- What if I used meaningful variable names?

```
def nextState(password, count):
    length = len(password)
    if (length >= 4):
        count += 1
        print(password)
        if (length >= 9):
            return password
    # FUNCTION CONTINUES...
```

Readability Example

- And replaced the magic numbers with constants?

```
def nextState(password, count):
    length = len(password)
    if (length >= 4):
        count += 1
        print(password)
        if (length >= 9):
            return password
    # FUNCTION CONTINUES...
```

Readability Example

- And replaced the magic numbers with constants?

```
def nextState(password, count):
    length = len(password)
    if (length >= MIN_LENGTH):
        count += 1
        print(password)
        if (length >= MAX_LENGTH):
            return password
    # FUNCTION CONTINUES...
```

Readability Example

- And added vertical space?

```
def nextState(password, count):
    length = len(password)
    if (length >= MIN_LENGTH):
        count += 1
        print(password)
        if (length >= MAX_LENGTH):
            return password
    # FUNCTION CONTINUES...
```

Readability Example

- And added vertical space?

```
def nextState(password, count):
    length = len(password)


    if (length >= MIN_LENGTH):
        count += 1
        print(password)


        if (length >= MAX_LENGTH):
            return password
    # FUNCTION CONTINUES...
```

Readability Example

- Maybe even some comments?

```
def nextState(password, count):
    length = len(password)


    if (length >= MIN_LENGTH):
        count += 1
        print(password)


        if (length >= MAX_LENGTH):
            return password
    # FUNCTION CONTINUES...
```

Readability Example

- Maybe even some comments?

```
def nextState(password, count):
    length = len(password)

    # if long enough, count as a password
    if (length >= MIN_LENGTH):
        count += 1
        print(password)

        # if max length, don't do any more
        if (length >= MAX_LENGTH):
            return password
    # FUNCTION CONTINUES...
```

Readability Example

- Now the purpose of the code is a bit clearer!
  - You can see how small, simple changes increase the readability of a piece of code

- This is actually part of a function that creates a list of the passwords for a swipe-based login system on an Android smart phone
  - Dr. Gibson wrote a paper on this, available here

Commenting

Commenting is an "Art"

- Though it may sound pretentious, it's true

- There are NO hard and fast rules for when a piece of code should be commented
  - Only guidelines
  - NOTE: This doesn't apply to **required** comments like file headers and function headers!

General Guidelines

- If you have a complex conditional, give a brief overview of what it accomplishes

```
# check if car fits customer criteria
if color == "black" and int(numDoors) > 2 \
        and float(price) < 27000:
```

- If you did something you think was clever, comment that piece of code
  - So that “future you” will understand it!

General Guidelines

- If you have a complex conditional, give a brief overview of what it accomplishes

```
# check if car fits customer criteria
if color == "black" and int(numDoors) > 2 \
        and float(price)
```

This backslash symbol tells Python that the code will continue on the next line.

- If you did something yo
comment that piece of code
  - So that "future you" will understand it!

General Guidelines

- **Don't** write obvious comments

```
# iterate over the list
for item in myList:
```

- **Don't** comment every line

```
# initialize the loop variable
choice = 1
# loop until user chooses 0
while choice != 0:
```

General Guidelines

- **Do** comment "blocks" of code

```
# calculate tip and total (if a party is
# large, set percentage to a minimum)
if (numGuests > LARGE_PARTY):
    percent = MIN_TIP
tip = bill * percent
total = bill + tip
```

General Guidelines

- **Do** comment nested loops and conditionals

```
listFib   = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
listPrime = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

# iterate over both lists, checking to see if each
# fibonacci number is also in the prime list
for num1 in listFib:
    for num2 in listPrime:
        if (num1 == num2):
            print(num1, "is both a prime and a \
                         Fibonacci number!")
```

General Guidelines

- **Do** comment very abbreviated variables names (especially those used for constants)
  - You can even put the comment at the end of the line!

```
MIN_CH    = 1      # minimum choice at menu
MAX_CH    = 5      # maximum choice at menu
MENU_EX   = 5      # menu choice to exit (stop)
P1_MARK   = "x"    # player 1's marker
P2_MARK   = "o"    # player 2's marker
```

“Good Code” – Adaptability

Adaptability

- Often, what a program is supposed to do evolves and changes as time goes on
  - Well-written flexible programs can be easily altered to do something new
  - Rigid, poorly written programs often take a lot of work to modify
- When coding, keep in mind that you might want to change or extend something later

Adaptability: Example

- Remember how we talked about not using "magic numbers" (or strings) in our code?

Bad:

```
def makeSquareGrid():
    temp = []
    for i in range(0, 10):
        temp.append([0] * 10)
    return temp
```

Good:

```
def makeSquareGrid():
    temp = []
    for i in range(0, GRID_SIZE):
        temp.append([0] * GRID_SIZE)
    return temp
```

0 and 1 are not "magic" numbers – why?

Adaptability: Example

- We can change **`makeSquareGrid()`** to be an even more flexible function

Better:

```
def makeSquareGrid(size):
    temp = []
    for i in range(0, size):
        temp.append([0] * size)
    return temp

# call makeSquareGrid
grid = makeSquareGrid(GRID_SIZE)
```

Good:

```
def makeSquareGrid():
    temp = []
    for i in range(0, GRID_SIZE):
        temp.append([0] * GRID_SIZE)
    return temp
```

Solving Problems

Simple Algorithms

- Input
  - What information we will be given, or will ask for
- Process
  - The steps we will take to reach our specific goal
- Output
  - The final product that we will produce

More Complicated Algorithms

- We can apply the same principles of input, process, output to more complicated algorithms and programs

- There may be multiple sets of input/output, and we may perform more than one process

Design Example

Questions when Designing

- What is the "big picture" problem?
- What sort of tasks do you need to handle?
  - What functions would you make?
  - How would they interact?
  - What does each function take in and return?
- What will your **`main()`** look like?

In-Class Example

- A program that allows two human players to play battleship, alternating turns
- Questions to consider:
- What do you want your board to look like? How do you want the user to play, or to select where they'll attack next?

In-Class Example

- A program that allows two human players to play battleship, alternating turns
- Design choices to consider:
  - What do you want your board to look like?
  - How do you want the user to play, or to select where they'll attack next?
  - How are you going to store the board?
  - What functions will you need?
  - What constants will you need?

Incremental Development

What is Incremental Development?

- Developing your program in small increments

1. Program a small piece of the program
2. Run and test your program
3. Ensure the recently written code works
4. Address any errors and fix any bugs
5. Return to step 1

Why Use Incremental Development?

- Incremental development:
  - Makes a large project more manageable
  - Leads to higher quality code
  - Makes it easier to find and correct errors
  - Is faster for large projects
    - May seem like you're taking longer since you test at each step, but faster in the long run

Debugging Woes

- Writing code is easy...
- Writing code that works correctly is HARD

- Sometimes the hardest part of debugging is finding out *where* the error is coming from
  - And solving it is the easy part (sometimes!)
- If you only wrote one function, you can start by looking there for the error

Announcements

- Project 2 out on Blackboard
  - Design due Friday, April 14th @ 8:59:59 PM
  - Project due Friday, April 21st @ 8:59:59 PM
  - Uses 3D lists and file I/O
- Final exam is when?
  - Friday, May 19th from 6 to 8 PM
- Survey #2 will be coming out soon