

CMSC201

Computer Science I for Majors

Lecture 15 – Program Design (cont)

Last Class We Covered

- The `range()` function
- Using `for` loops
 - Using `for` loops and `range()`
 - Difference between `for` and `while` loops

Any Questions from Last Time?

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 *very, very, **very*** 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 horizontal 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 horizontal 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

- 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 meaningful comments?

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

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

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

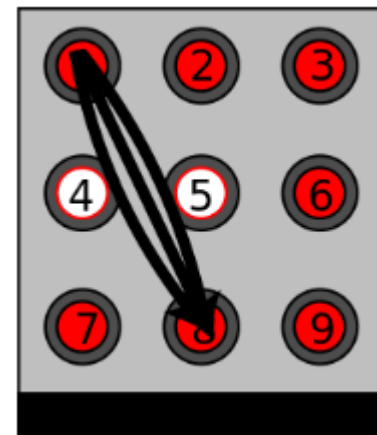
Readability Example

- Maybe even some meaningful 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 possible passwords for a swipe-based login system on an Android smart phone
 - Dr. Gibson co-wrote a paper on this, available [here](#)



Commenting

Commenting is an “Art”

- Though it sounds 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

- **Don't** write obvious comments

```
# iterate over the list
for i in range(len(myList)):
```

- **Don't** comment every line

```
# initialize the loop variable
choice = 1
# loop until user chooses to quit
while choice != QUIT:
```

General Guidelines

- Do comment “blocks” of code

```
# calc tip and total (set min for large parties)
percent = float(input("Enter tip percent: "))
if numGuests > LARGE_PARTY and percent < MIN_TIP:
    percent = MIN_TIP
    print("There is a minimum tip of", MIN_TIP, \
          "for large parties")

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]
```

```
# check to see if each fibonacci number
# is also in the prime number list
for f in range(len(listFib)):
    for p in range(len(listPrime)):
        if (listFib[f] == listPrime[p]):
            print(listFib[f], "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!
As long as the comment won't wrap around

```
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

- Here is an example of a poorly modular function

Bad:

```
def makeSquareGrid() :  
    grid = []  
    row = []  
    for i in range(10) :  
        row.append(0)  
    for i in range(10) :  
        grid.append( row[:] )  
    return grid
```

How can we improve this function to be more modular and adaptable?

Adaptability: Example

- Let's make the size of the grid a parameter

Good:

```
def makeSquareGrid(size):  
    grid = []  
    row = []  
    for i in range(size):  
        row.append(0)  
    for i in range(size):  
        grid.append( row[:] )  
    return grid
```

Adaptability: Example

- And let's add the element as a parameter too

Better:

```
def makeSquareGrid(size, elem):  
    grid = []  
    row = []  
    for i in range(size):  
        row.append(elem)  
    for i in range(size):  
        grid.append( row[:] )  
    return grid
```

How could we
adjust this to allow
non-square grids?

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?

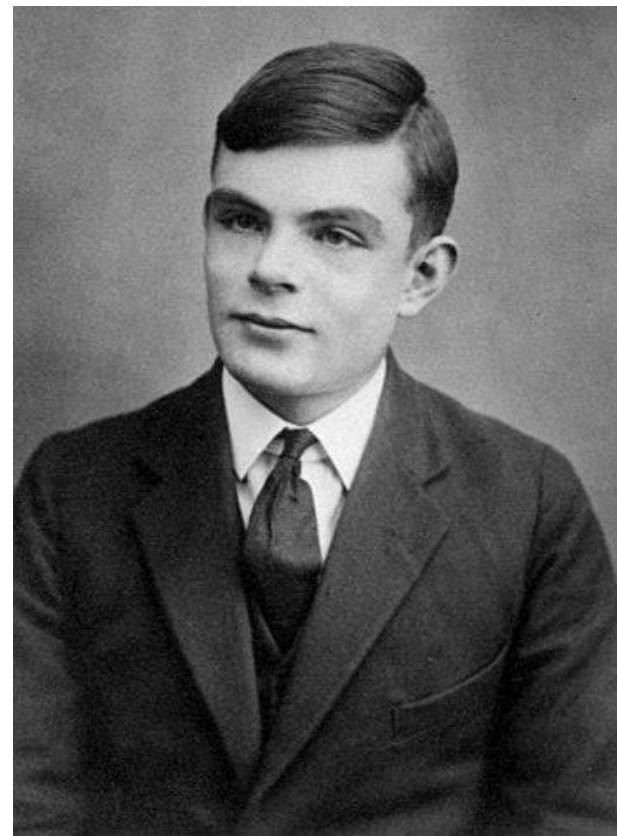
- Incremental development:
 - Makes a large project more manageable
 - Leads to higher quality code
 - Makes it easier to find and correct errors
 - Is faster for large projects
 - May seem like you're taking longer since you test at each step, but faster in the long run

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 since the last run, start by looking there for the error

Daily CS History

- Alan Turing
 - Helped to break the German Engima cipher during WWII
 - Proposed the “Turing test” to measure artificial intelligence
 - Turing “machines”
 - Designed the first computer chess program in 1953
 - Talented long-distance runner



Announcement: Advising

- CMSC and CMPE students, sign up for an advising appointment.
 - <http://advising.coeit.umbc.edu/registration/>
- Select that you are in MATH 150 or higher and haven't completed the gateway.
- There are both group advising and individual advising appointments open. The earliest dates available are for group advising.

Announcements

- Project 2 is out on Blackboard now
 - Design is due by Friday (Nov 2nd) at 8:59:59 PM
 - Project is due by Friday (Nov 9th) at 8:59:59 PM
- Midterm #2 is coming up
 - November 14th and 15th in class

Image Sources

- Android password swipe:
 - http://static.usenix.org/events/woot10/tech/full_papers/Aviv.pdf
- Alan Turing:
 - https://en.wikipedia.org/wiki/File:Alan_Turing_Aged_16.jpg