

CMSC201

Computer Science I for Majors

Lecture 15 – Program Design

Prof. Katherine Gibson

Last Class We Covered

- Functions
 - Returning values
 - Returning multiple values at once
- Modifying parameters
 - Mutable
 - Immutable
- Modular programming

Any Questions from Last Time?

Today's Objectives

- To discuss the details of “good code”
- To learn how to design a program
- How to break it down into smaller pieces
 - Top Down Design
- To introduce two methods of implementation
- To learn more about Modular Development

“Good Code” – Readability

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 this
 - Readability
 - Adaptability

Readability

- Having your code be readable is important, both for your sanity and someone else's
- 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
 - 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, c  
    # FUNCTION CONTINUES...
```

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

Readability Example

- What if I added meaningful variable names?

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

Readability Example

- What if I added meaningful variable names?

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

```
    # FUNCTION CONTINUES...
```

Readability Example

- Now the purpose of the code is a bit clearer!
 - (It's actually part of some code that generates a complete list of the possible passwords for a swipe-based login system on a smart phone)
- You can see how small, simple changes increase the readability of a piece of code

Commenting is an “Art”

- Though it may sound pretentious, it’s true
- There are NO hard and fast rules for when to a piece of code should be commented
 - Only guidelines
 - (This doesn’t apply to required comments like file headers, though!)

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 int(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 item in myList:
```

- **Don't** comment every line

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

“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” in our code?

Bad:

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

Good:

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


Adaptability: Example

- In the whole of this program we use **GRID_SIZE** a dozen times or more
 - What if we suddenly want a bigger or smaller grid? Or a variable sized grid?
 - If we've left it as 10, it's very hard to change
- But **GRID_SIZE** is very easy to change
 - Our program is more adaptable

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

Complex Problems

- If we only take a problem in one piece, it may seem too complicated to even begin to solve
 - Creating your own word processor
 - Making a video game from scratch
 - A program that recommends classes based on availability, how often the class is offered, and the professor's rating

Top Down Design

Top Down Design

- Computer programmers use a divide and conquer approach to problem solving:
 - Break the problem into parts
 - Solve each part individually
 - Assemble into the larger solution
- These techniques are known as *top down design* and *modular development*

Top Down Design

- Breaking the problem down into pieces makes it more manageable to solve
- *Top-down design* is a process in which a big problem is broken down into small sub-problems, which can themselves be broken down into even smaller sub-problems

Top Down Design: Illustration

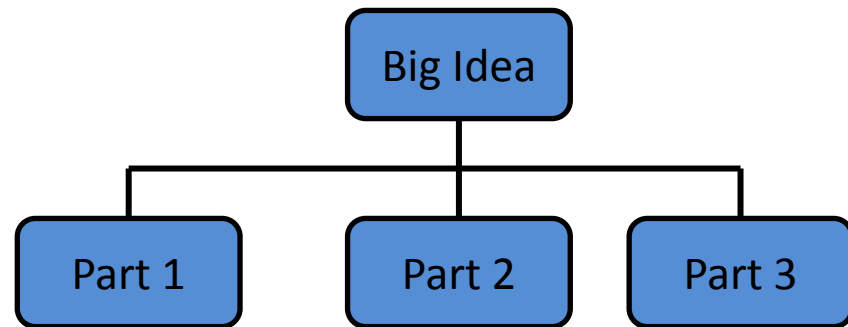
- First, start with a clear statement of the problem or concept
- A single big idea



Big Idea

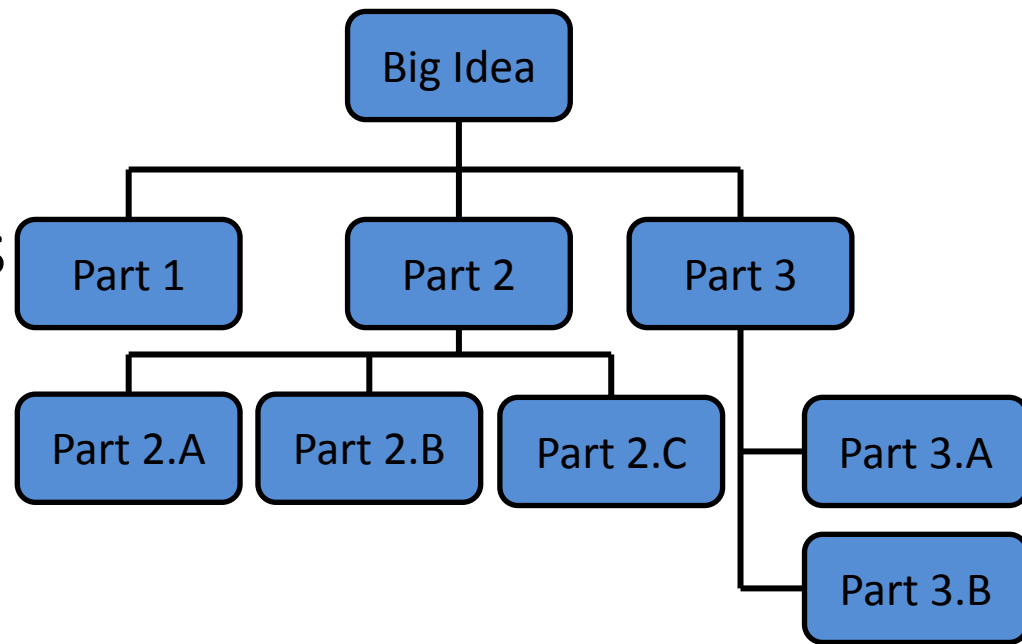
Top Down Design: Illustration

- Next, break it down into several parts



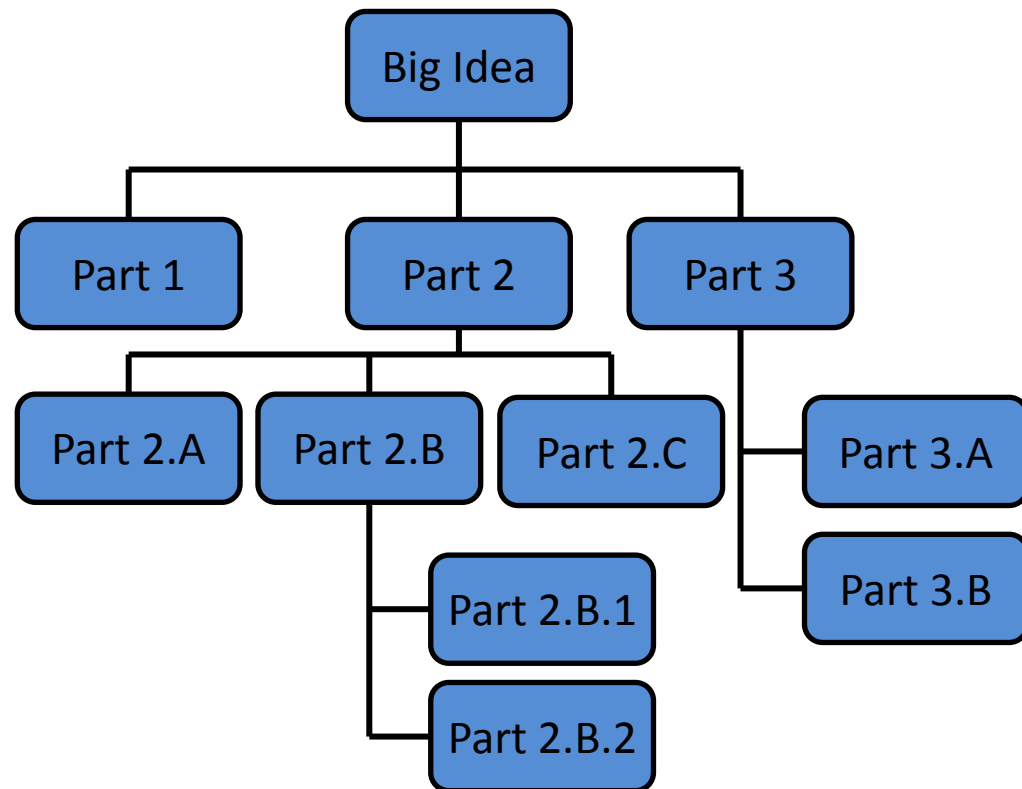
Top Down Design: Illustration

- Next, break it down into several parts
- If any of those parts can be further broken down, then the process continues...



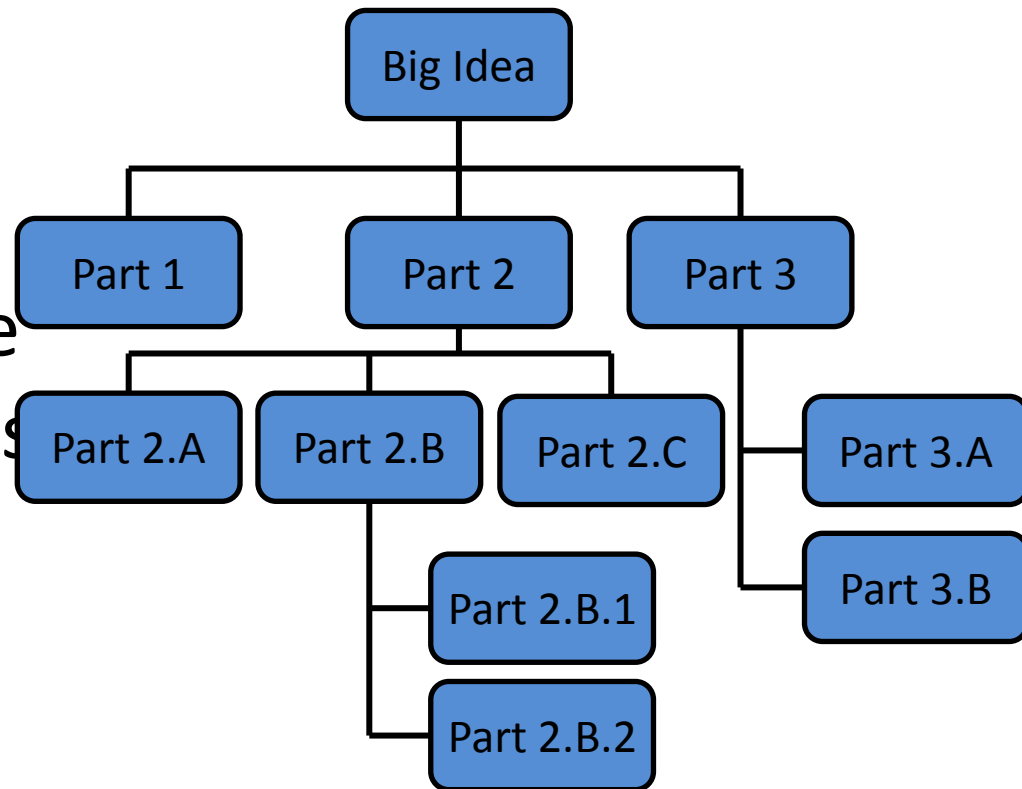
Top Down Design: Illustration

- And so on...



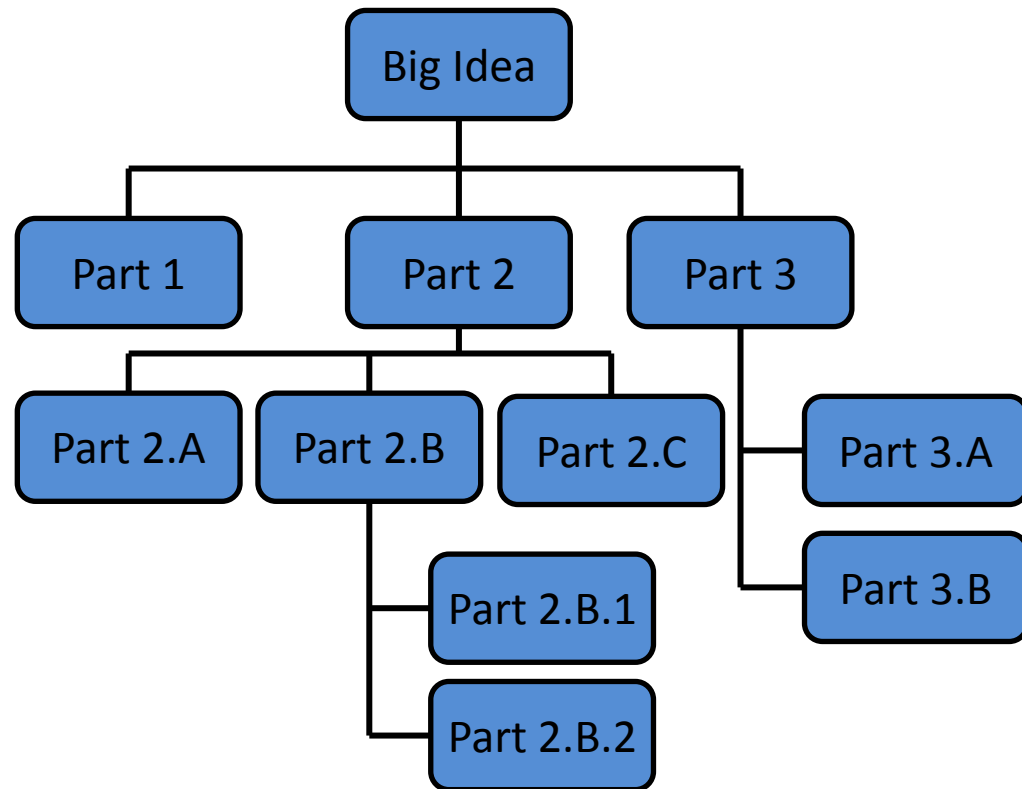
Top Down Design: Illustration

- Your final design might look like this chart, which shows the overall structure of the smaller pieces that together make up the “big idea” of the program



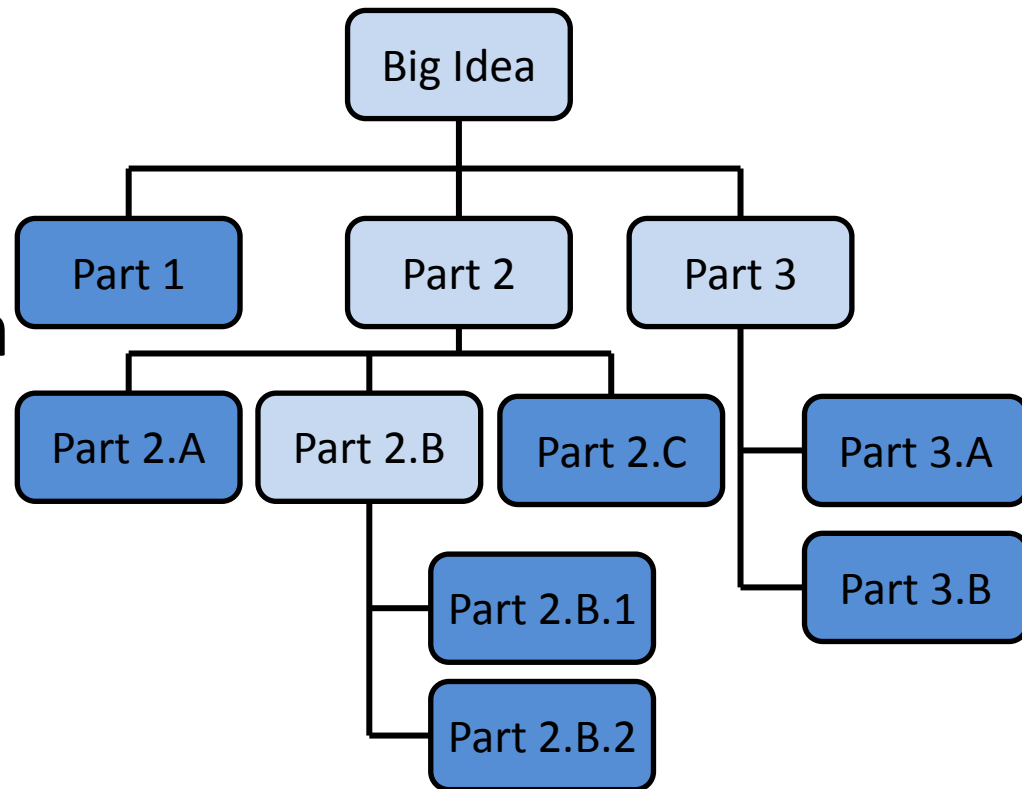
Top Down Design: Illustration

- This is like an upside-down tree, where each of the nodes represents a process



Top Down Design: Illustration

- The bottom nodes represent pieces that need to be developed and then recombined to create the overall solution to the original problem.



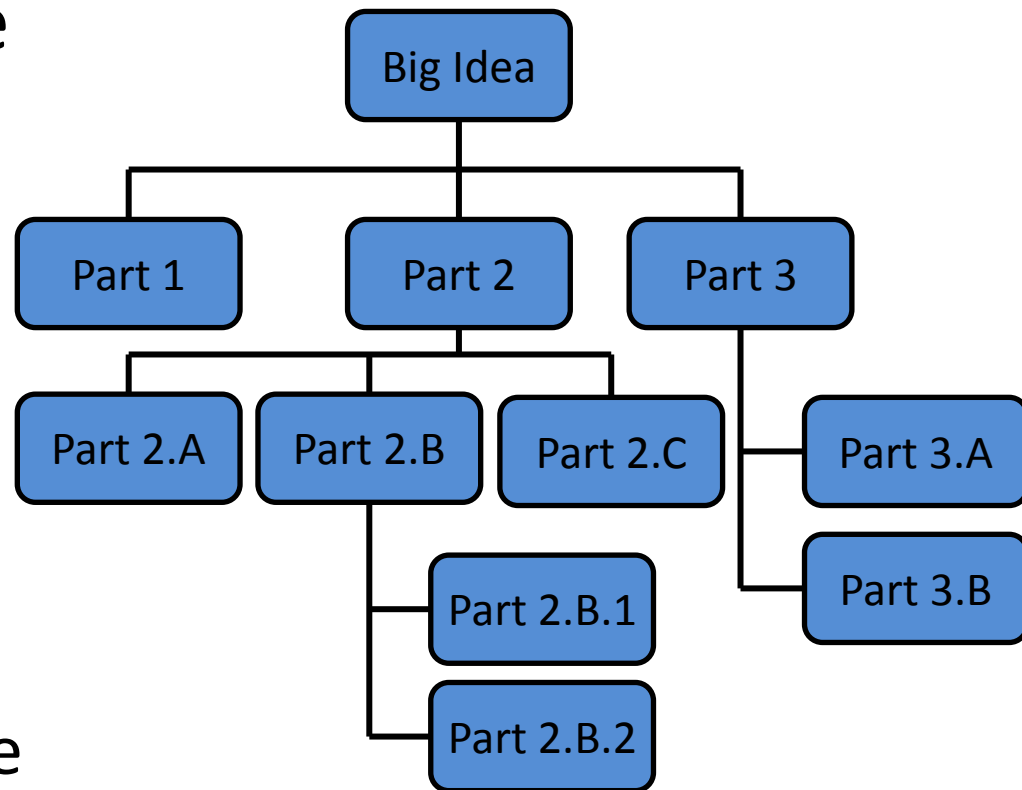
Analogy: Paper Outline

- Think of it as an outline for a paper you're writing for a class assignment
- You don't just start writing things down
 - You come up with a plan of the important points you'll cover, and in what order
 - This helps you to formulate your thoughts as well

Implementing from a Top Down Design

Bottom Up Implementation

- Develop each of the modules separately
 - Test that each one works as expected
- Then combine into their larger parts
 - Continue until the program is complete



Bottom Up Implementation

- To test your functions, you will probably use `main()` as a (temporary) testing bed
- Calling functions with different test inputs
 - Ensuring that functions “play nicely” together

Top Down Implementation

- Create “dummy” functions that fulfill the requirements, but don’t perform their job
 - For example, a function that is supposed to take in a file name and return the weighted grades simply returns a 1
- Write up a “functional” `main()` that calls these dummy functions
 - Help pinpoint other functions you may need

How To Implement?

- Top down? Or bottom up?
- It's up to you!
 - As you do more programming, you will develop your own preference and style
- For now, just use something – don't code up everything at once without testing anything!

In-Class Example

In-Class Example

- (Expanding on the “Used Car Lot” from Lab 8)
- You run a Used Car Lot franchise, with multiple locations in the area
 - Every morning you get a list of available cars from each location as a separate file
 - Customers may come in and request any combination of features (color, price, etc.)
 - You have to handle your stock for the day, and handle customers who ask for impossible things

In-Class Example

- 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

- Specifics:
 - Keep track of what cars are available at each location, and which have already been sold
 - Read in stock at beginning of program (“morning”)
 - Write down stock at end of the program (“closing shop”)
 - Don’t accept requests for things like 8 door cars
 - Customers don’t need a preference for everything
 - *e.g.*, a 4 door under \$35,000 – but don’t care what color
 - Offer the option to buy from another location

Modular Development

Why Use Modular Development?

- Modular development of computer software:
 - makes a large project more manageable
 - is faster for large projects
 - leads to a higher quality product
 - makes it easier to find and correct errors
 - increases the reusability of solutions

Managing Large Projects

- Makes a large project more manageable...
- Easier to understand tasks that are smaller and less complex
- Smaller tasks are less demanding of resources

Faster Project Development

- Is faster for large projects...
- Different people work on different modules
- Then put their work together

- Different modules developed at the same time
 - Speeds up the overall project

Higher Quality Product

- Leads to a higher quality product...
- Assign people to use their strengths
- Programmers with knowledge and skills in a specific area, such as graphics, accounting, or data communications, can be assigned to the parts of the project that require those skills

Correcting Errors

- Makes it easier to find and correct errors...
- Sometimes the hardest part of debugging is finding out *where* the error is coming from
 - And solving it is the easy part
 - (Sometimes!)
- Modular development makes it easier to isolate the part of the software that is causing trouble

Reuse of Code (Solutions)

- Increases the reusability of solutions...
- Solutions to small, targeted problems are more likely to be useful elsewhere than solutions to bigger problems
 - *e.g.*, getting valid user input (returns one int)
vs. getting and calculating quiz grades
- They are more likely to be reusable code

Libraries

- Over time, you may develop your own “library” of useful functions
- Just like Python has libraries for doing things with strings, opening and writing to files, and other common tasks you might want to do

Final In-Class Exercise

- What functions would you need to write a tic-tac-toe program that plays from the terminal?
- How would they interact?
- Draw a diagram if you need to!

Any Other Questions?

Announcements

- We'll go over the exam in class next time
 - Bring your exam with you!
- Homework 7 is out
 - Due by Thursday (Oct 29nd) at 8:59:59 PM
- Project 1 will be out Oct 29th