

Final Exam Review

15-110 – Friday 4/24

Announcements

- Congratulations on finishing Hw6!
- Reminder: Final Exam is Monday 04/27 8:30-11:30am
- See Piazza post for details:
 - <https://piazza.com/class/mk3373pfoq3k/post/110>

Feedback Time

- We appreciate your feedback on how this class went.
 - FCEs for instructors & the class in general: <https://cmu.smartevals.com>
 - SCS form for TAs: <https://www.ugrad.cs.cmu.edu/ta/S26/feedback/>
 - 15-110 Post-Semester Survey: <https://bit.ly/110-s26-post>
 - Fill this out by Friday 05/01 at noon to get 1 bonus point on Hw6
- You do: take five minutes now to fill out feedback forms for this class



Learning Goals

- Review topics from units 3, 4 and 5:
 - **Simulation**
 - **Data Analysis**
 - **Managing Large Code Projects**
- Note:
 - Units 1 and 2 will be **included** in the final exam
 - Today we will cover a **subset** of Units 3-5 (not everything)

Simulation

What are simulations?

A **simulation** is an automated imitation of a real-world event.

Simulations share a lot in common with real world experiments. Major differences include:

- Experiments run in **real time**; simulations can be **sped up, slowed down, or paused**.
- Experiments can be **expensive**; simulations are fairly **cheap**.
- Experiments include **all possible factors**; simulations only include **factors we program in**.

Simulations Run on Models

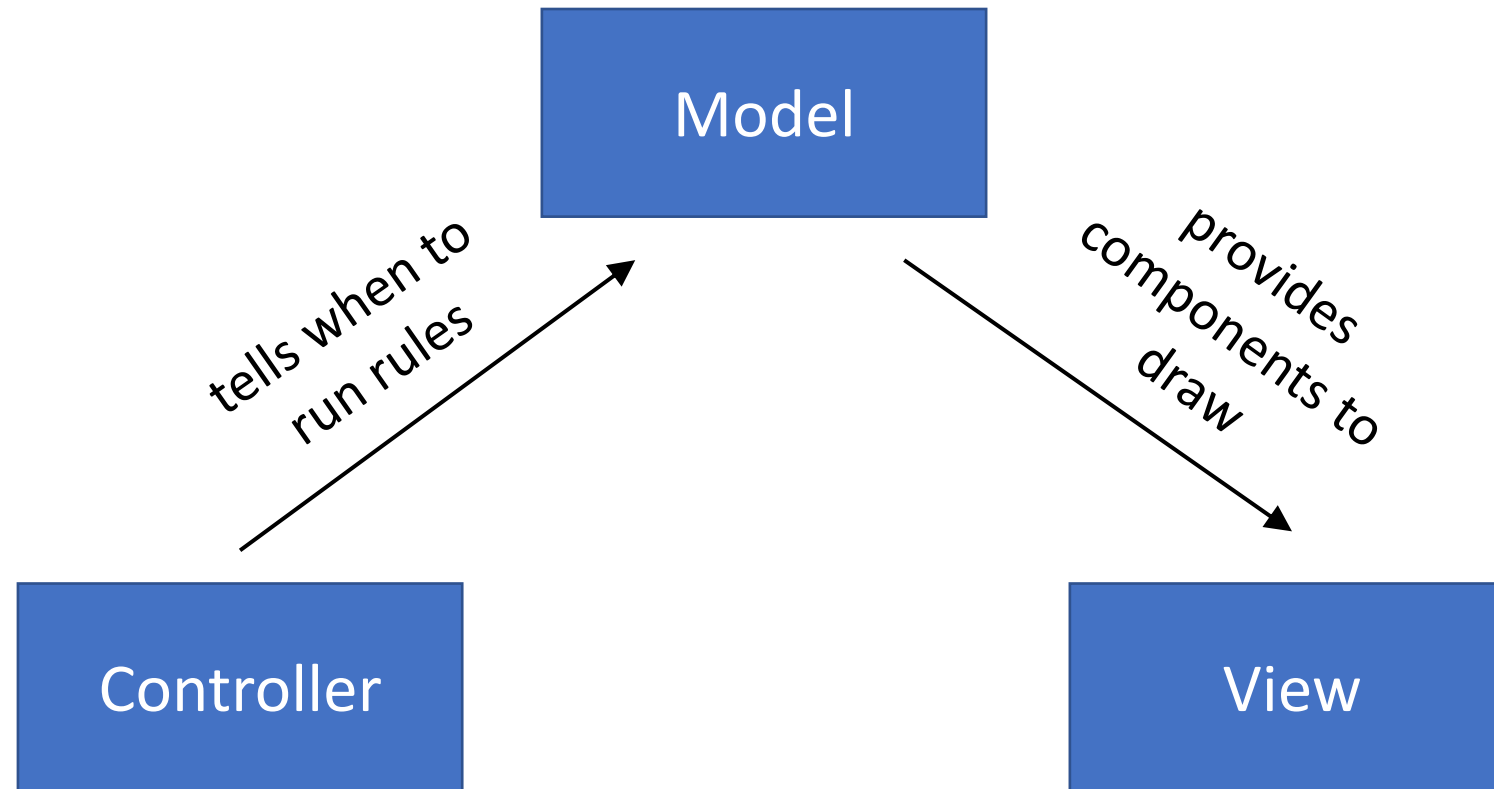
How do we program a simulation? You need to design a good **model**, which will mimic the part of the real world you want to study. The simulation showcases how the system represented by the model changes **over time**, or how it changes **based on events**.

Models are composed of two parts:

- The **components** of the system (information that describes the world at an exact moment).
- The **rules** of the system (how the components should change as time passes/events occur).

Components are like variables, and rules are like functions!

Model, View, Controller



You do:

You are given the following simulation code which runs in a 400x400 window.

- What does the model of the world represent?
- What is the view doing?
- What kind of simulations is it (time or event based?)
- Describe in your own words what the simulation is doing.

Now, change this simulation so that the color of the circle changes red to blue to red to blue as time goes by!

```
def makeModel(data):  
    data["r"] = 100  
  
def makeView(data, canvas):  
    canvas.create_oval(  
        200 - data["r"], 200 - data["r"],  
        200 + data["r"], 200 + data["r"])  
  
def keyPressed(data, event):  
    if event.keysym == "Up":  
        data["r"] += 10  
    elif event.keysym == "Down":  
        data["r"] -= 10
```

Solution

```
def makeModel(data):  
    data["r"] = 100  
    data["c"] = "red"  
  
def makeView(data, canvas):  
    canvas.create_oval(  
        200 - data["r"], 200 - data["r"],  
        200 + data["r"], 200 + data["r"]),  
        fill=data["color"])  
  
def runRules(data, call):  
    if data["color"] == "red":  
        data["color"] = "blue"  
    else:  
        data["color"] = "red"
```

Data Analysis

Data Analysis Gains Insights on Data

Data Analysis is the process of using computational or statistical methods to **gain insights** about data.

Data Analysis is used widely by many organizations to answer questions in many different domains. It plays a role in everything from advertising and fraud detection to airplane routing and political campaigns.

Data Analysis is also used widely in **logistics**, to determine how many people and how much stock is needed and where they should go.

Three Types of Data

When we work with simple data, that data often falls into one of three types. These types will determine what analyses we run.

Categorical: Data fall into one of several categories. Those categories are separate and cannot be compared.

Example: style of house (ranch, split-level, two-story, duplex, Victorian, etc.)

Ordinal: Data fall into separate categories, but those categories **can** be compared – they have a specific **order**.

Example: what is the condition of the house? (poor, fair, good, excellent, new)

Numerical: Data are **numbers**. We can perform mathematical operations on it and compare it to other data.

Example: how large is the house in square feet?

Reference Tables for Visualizing Data

One-Dimensional	
Numerical	Histogram
Ordinal	Bar Chart
Categorical	Pie Chart

Two-Dimensional	Numerical	Ordinal/Categorical
Numerical	Scatter Plot	Bar Chart or Box-and-Whiskers Plot
Ordinal/Categorical	Bar Chart or Box-and-Whiskers Plot	Table

Three-Dimensional	Numerical	Ordinal/Categorical
Numerical	x Numerical: Bubble Plot or Scatter Plot Matrix x Ordinal/Categorical: Colored Scatter Plot	x Numerical: Colored Scatter Plot
Ordinal/Categorical	x Numerical: Colored Scatter Plot	Table

Tools to Use

Once you've identified where the information is located, use **string slicing** and **string methods** to separate out the information you need.

Slicing (`s[start:end:step]`) can be used to remove parts of the data that are unnecessary.

The **split** method (`s.split(".")`) can be used to break up data that is separated by a known delimiter.

The **index** method (`s.index(":")`) can be used to find the location of the beginning or end of a section. That can be combined with slicing or splitting to isolate the needed data.

The **strip** method (`s.strip()`) can be used to remove whitespace (spaces, tabs, and newlines) from the front and back of a string. This is useful for isolating the core text of a string.

You do:

You have data about some books you have bought. The data is a 2D list and includes the title of the book, the genre, the number of pages and the price of the book, all as **strings**.

```
books = [ ["title", "genre", "pages", "price"],  
  ["The Hobbit", "fantasy", "310 pages", "$12.99"], ["1984", "dystopian", "328 pages", "$15.50"],  
  ["Dune", "sci-fi", "688 pages", "$11.99"], ["The Name of the Wind", "fantasy", "662 pages", "$16.99"],  
  ["Neuromancer", "sci-fi", "271 pages", "$18.75"], ["Mistborn", "fantasy", "295 pages", "$11.25"] ]
```

Write the function `findBooks(books, genre, pages, price)` which takes a 2D list, a string of a genre, an integer for number of pages and a float for price. The function returns a list of book titles in that genre with a number of pages less than the given pages value and a price less than the given price value. For example:

```
findBooks(books, "fantasy", 350, 13.00) → ["The Hobbit", "Mistborn"] # as we have bolded above  
findBooks(books, "sci-fi", 300, 15.00) → [ ] # no sci-fi books < 300 pages and < $15.00
```

Hint: you'll need to do some data processing. Think about how to skip the header and how to turn the number of pages and price into numbers. **The data will always be formatted as shown above**, though different entries may be used after the header.

Solution

```
def findBooks(books, genre, pages, price):  
    result = [ ]  
    for entry in books:  
        if entry[1] == genre:  
            bookPages = int(entry[2].split(" ")[0])  
            bookPrice = float(entry[3][1:])  
            if bookPages < pages and bookPrice < price:  
                result.append(entry[0])  
    return result
```

Managing Large Code Projects

Helper Functions

We refer to a function that solves a subpart of a larger problem as a **helper function**. By breaking up a large problem into multiple smaller problems and solving those problems with helper functions, we can make complicated tasks more approachable.

Try to identify **subtasks** that are repeated or are separate from the main goal; break down the problem into smaller parts. Have **one subtask per function** to keep things simple.

Helper Functions



We want to implement a text-based game of Connect Four. It is a two-player game where players take turns putting pieces on a two-dimensional grid, and whoever gets four pieces in a row first wins the game.

Assume we already wrote four helper functions for this game:

- `makeBoard()`, takes no input and returns a 2D list representation of the board
- `displayBoard(board)`, takes the 2D list representation of the board and displays it
- `makeMove(board, player)`, takes the 2D list representation of the board and the current player (either "red" or "blue"), gets a valid move from the player, applies that move to the board, and returns the updated board
- `isGameOver(board)`, takes the 2D list representation of the board and returns True if the game is over and False otherwise

Write the main game function!

Solution

```
def connectFour():  
    b = makeBoard()  
    displayBoard(b)  
    currentPlayer = "red"  
    while isGameOver(b) == False:  
        b = makeMove(b, currentPlayer)  
        displayBoard(b)  
        if currentPlayer == "red":  
            currentPlayer = "blue"  
        else:  
            currentPlayer = "red"
```

Learning Goals

- Review main topics from units 3, 4 and 5:
 - **Simulation**
 - **Data Analysis**
 - **Managing Large Code Projects**
- Note:
 - Units 1 and 2 will be **included** in the final exam
 - Today we will cover a **subset** of Units 3-5 (not everything)