Thinking Like A Computer

Skills to help you outline and (eventually) debug code

Andie M. Creel

Thanks to Ethan Addicott and Matt Gordon for early iterations of this lecture

My background

- Freshman in college: financial engineering -> Java
  - “I am a magician”
- Majored in Econ, minored in CS
- TAed programming labs in college
- Pandemic Hobby: big data
- Master’s thesis: Parks use in pandemic (Cell Phone Data)
- Dissertation work: Environmental Econ -- value of local recreation and urban green space

My motivation for knowing how to program

- Sustainable development: "meets the needs of the present without compromising the ability of future generations to meet their own needs" -- Brundtland Report, 1987
- Needs: consumption of goods, health, political stability, culture
  - The environment affects this
  - That effect can be hard to observe/measure
- The explosion of data and computing power can help us observe and understand how nature affects sustainable development
- "What gets measured gets managed" – Peter Drucker (maybe?)

Goal of my Foundation Lectures

- Get rid of the intimidation factor
- Build a foundation for opportunities to build on
- Accelerate the initial learning curve
  - I tried to build what I wish I had

- Advice: Ask questions
  - Believe that you're entitled to learn everything in this certificate program
  - Even when you don't know how to say it, try to ask it

What is programming

1. Bits and bytes – what the computer actually speaks (0s and 1s)
2. Machine and Assembly languages -- translates our code to 0s and 1s
3. Code – R, Python, etc.
4. Software – Windows, iOS, Applications, Excel, R Studio

What is programming

1. Bits and bytes – what the computer actually speaks (0s and 1s)
2. Machine and Assembly languages -- translates our code to 0s and 1s
3. **Code – R, Python, etc.**
4. Software – Windows, iOS, Applications, Excel, R Studio

Some definitions

- Code: What your computer runs (different languages)
- Script: The text file where you write your code
- Comments: Notes you write yourself that the computer doesn’t read
- Run (aka compile and execute): when the computer executes your code
- IDE/GUI/Software
  - Where you write and run your scripts and comments
  - R Studio, Jupyter Notebooks, Google Collab (Collaboratory)

Why should you think like a computer?

- Clear definition of “problems”
  - Cannot solve a problem that isn’t defined (how do you know it's solved?)
  - Forces precision and accuracy in problem definition
  - Collaboration: Once clear to you, can also be clear to team members
- Solutions to “problems”
  - Environmental field is filled with giant problems
  - To make progress, need to break them into solvable pieces
  - Learning to program is learning how to break big problems down
  - Steps are clear

Why should you learn to code?

- **Efficiency and Speed:** it’s so much faster
- **Accurate:** not reliant on copy-paste/find-replace
- **Replicability:** teams, peer review, new datasets
- **Customizable:** write models for your data, clean and manipulate data for your models
- **Employment:** more jobs, better jobs, more money
- **Collaboration:** compared to Excel, later in certificate we’ll discuss Github

Pseudo Code – first step of coding

- What
  - Pseudo code is the outline of your code
  - Similar to the writing process
  - Not a coding language
  - I do it on scrap paper or in comments at the top of my scripts
- Why
  - Clarify your logic
  - Serves as road map and plan
    - Super important for multiple work sessions
  - Collaboration/Documentation
    - Can communicate what you've done/are doing to others

Exercise One: Coffee

Write “instructions” for how to brew a cup of coffee/tea

[pause video for 3 min to do so]

Exercise One: Coffee

My Pseudo Code

Step 1: Fill coffee machine with water
Step 2: Turn on machine to boil water
Step 3: While waiting for water to boil
    a. Place coffee filter in coffee maker
    b. Add coffee grounds to filter
Step 4: Wait for coffee to drip through the filter
Step 5: Remove filter and discard used grounds
Step 6: Pour coffee into a mug
Step 7: Add any desired milk or sugar
Step 8: Stir and enjoy

Exercise One: Coffee

Take Aways

- Is your pseudo code the same as mine?
  - Probably not!
  - And that’s good!
- A critical part is the *problem definition* (*i.e.,* the goal)
- Many solution strategies could work
- Both you and I outlined steps to achieve the well-defined goal of brewing a cup of coffee
  - We each solved the problem
  - But solution strategy was different

Exercise Two: Rock Paper Scissors

Write instructions for how to play Rock, Paper, Scissors

[pause video for 5 min to do so]

My pseudo code (lazy)

Player: choose rock paper scissor

Computer: random generate rock paper scissor

Compare P & C

- If P = Rock
  - And C = Rock: tie
  - And C = paper: player loses
  - And C = scissors: player wins
- Same for if P = paper and P = Scissors

Not Lazy Pseudo Code

```
While play_again is true
    Prompt the player to select Rock, Paper, or Scissors
    Generate a random choice for the computer (Rock, Paper, or Scissors)

    If player's choice is the same as computer's choice
      Display "It's a tie!"
    Else If player chooses Rock and computer chooses Scissors
      Display "Player wins! Rock crushes Scissors."
    Else If player chooses Paper and computer chooses Rock
      Display "Player wins! Paper covers Rock."
    Else If player chooses Scissors and computer chooses Paper
      Display "Player wins! Scissors cut Paper."
    Else
      Display "Computer wins!"

    Ask the player if they want to play again
       set play_again to true or false
```

R Code

```r
# Function to get computer's choice
get_computer_choice <- function() {
  choices <- c("Rock", "Paper", "Scissors")
  sample(choices, size = 1)
}

# Function to determine the winner
determine_winner <- function(player_choice, computer_choice) {
  if (player_choice == computer_choice) {
    return("It's a tie!")
  } else if (player_choice == "Rock" && computer_choice == "Scissors") {
    return("Player wins! Rock crushes Scissors.")
  } else if (player_choice == "Paper" && computer_choice == "Rock") {
    return("Player wins! Paper covers Rock.")
  } else if (player_choice == "Scissors" && computer_choice == "Paper") {
    return("Player wins! Scissors cut Paper.")
  } else {
    return("Computer wins!")
  }
}
```

```r
while (play_again) {
  # Get player's choice
  player_choice <- readline(prompt = "Choose Rock, Paper, or Scissors:

  # Validate input
  while(!(player_choice %in% c("Rock", "Paper", "Scissors"))) {
    player_choice <- readline(prompt = "Invalid choice. Choose Rock, Pa
  }

  # Get computer's choice
  computer_choice <- get_computer_choice()
  cat("Computer chose:", computer_choice, "\n")

  # Determine and display the winner
  result <- determine_winner(player_choice, computer_choice)
  cat(result, "\n")

  # Ask if the player wants to play again
  play_again_input <- readline(prompt = "Play again? (yes/no): ")
  play_again <- tolower(play_again_input) == "yes"
}

cat("Game Over. Thanks for playing!")
```

Couple of notes on writing, debugging code

- “I minored in stack overflow”
- I still have to look things up constantly
- Knowing programming makes me better at:
  - Defining problems
  - Developing solution strategies
  - Solving harder and harder problems
- Documentation, Google, Stack Overflow, and ChatGPT can help you from there
- But if you don’t know how to define and solve problems, you won’t be able to take full advantage of these aids

Couple of notes on writing, debugging code

- “I minored in stack overflow”
- I still have to look things up constantly
- Knowing programming makes me better at:
  - Defining problems
  - Developing solution strategies
  - Solving harder and harder problems
- Documentation, Google, Stack Overflow, and ChatGPT can help you from there
- But if you don’t know how to define and solve problems, you won’t be able to take full advantage of these aids

ChatGPT and other AI tools

- Any of them could probably solve all of what we do in this class
- But they can’t solve everything you’ll eventually want to do
- You can’t get to the cutting-edge problems without basics
  - At the very least, you’ll be limited
- Learn basics so that you can get to hard problems, and use AI as a collaborative aid once there