As you arrive:

1. Start up your computer and plug it in
2. ***Log into Angel*** and go to CSSE 120
3. Do the ***Attendance Widget*** – the PIN is on the board
4. Go to the course ***Schedule Page***
5. Open the ***Slides*** for today if you wish
6. Check out today's project: `Session14_NestedLoops`

Plus in-class time working on and practicing these AND other concepts.

Top Down Design

- The BlackJack game, e.g., Part 1
- Designing a larger program
- Top-down design

More Compute-in-a-Loop Patterns

- Nested Loops
  - For, While
- Wait-for-event Loops

Outline of Today's Session

Checkout today's project:
`Session14_NestedLoops`

- Exam 1 Redux
- Questions?
- How to design a larger program
  - Top-down design
    - What is it?
    - An example of doing it: the BlackJack program
- Compute-in-a-Loop Patterns:
  - Review: for, while (interactive, sentinel), loop-and-a-half (sentinel), file
  - Nested Loops
  - The Wait-For-Event compute-in-a-loop Pattern

Team preference survey

- Beginning with Session 15, you will be working on a team project.
- This survey is a chance for you to tell us your preferences for who you want to work with.
  - Also has questions about your "work style" to help us form teams.
  - Suggestion: prefer people whose understanding level is similar to yours.
  - Fill out the survey, even if you have no preference.
- **Due before 11:59 p.m. TONIGHT.**

Designing/implementing a larger program

- Until now, our programs have been small and simple
  - Possible exceptions: pizzPolyStar, speedReading
- For larger programs, we need a strategy to help us be organized
- One common strategy: **top-down design**
  - Break the problem into a few big pieces (functions)
  - Break each piece into smaller pieces
  - Eventually we get down to manageable pieces that do the details

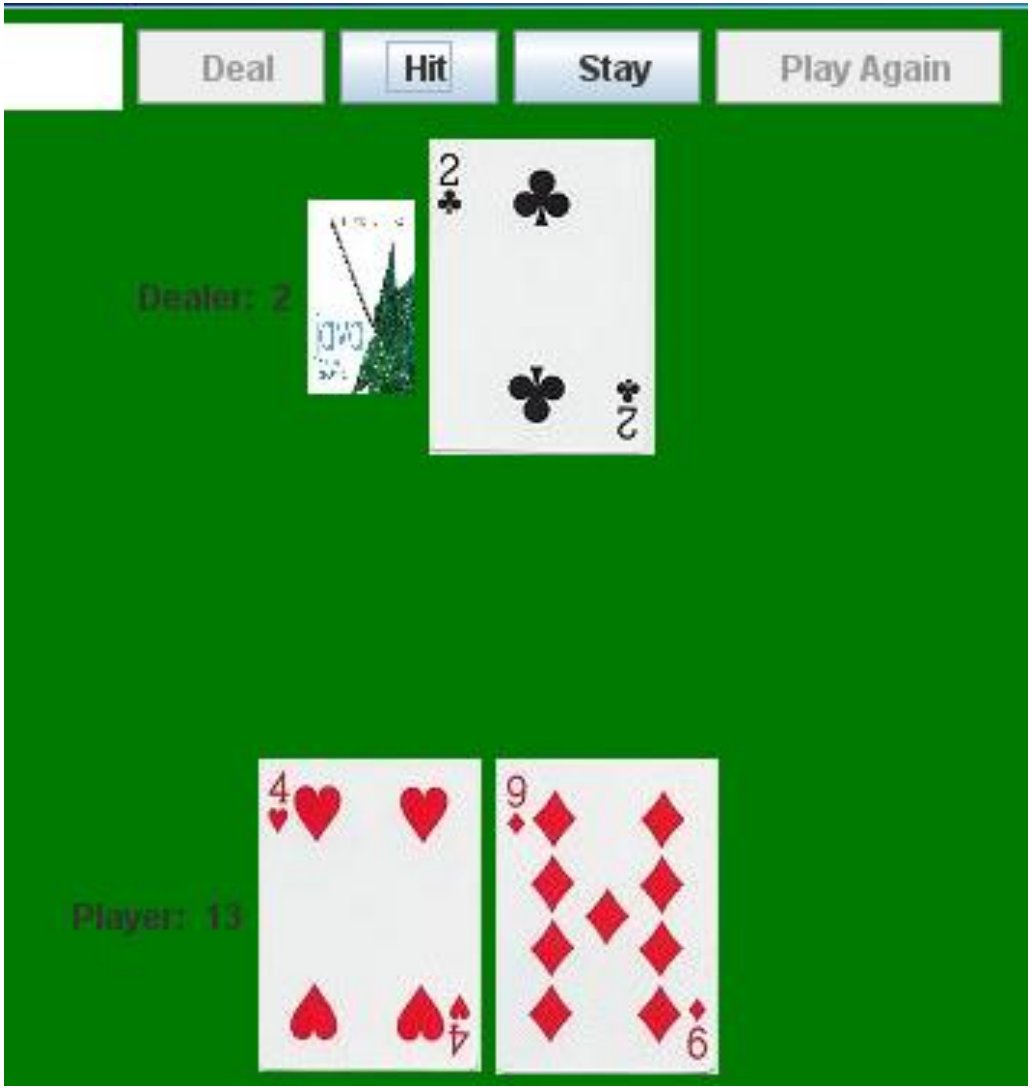
Example: Two-player blackjack (21)

- Uses a regular deck of cards
- Player and Dealer each initially get two cards
- Player can see both of own cards, but only one of dealer's cards
- Suit is irrelevant, only denomination determines points per card:
  - Ace: one point or 11 points.
  - 2-10: point value is the number of the card.
  - face card: 10 points
- Object: Get as close as you can to 21 points in your hand without going over

Q3a

Blackjack illustration

- We won't develop a GUI today, but this image from a GUI Blackjack game* illustrates how the game goes



- * from Lewis and Chase, *Java Software Structures*

Blackjack play

- Player has the option to take one or more "hits" (cards) or to "stay" (keep the current hand)
- If a hit increases the Player's score to more than 21, (s)he is "busted" and loses immediately
- If the Player is not busted, the Dealer plays, but with more constraints
  - If the Dealer's score is less than 16, (s)he must take a hit
  - Otherwise, (s)he must stay
- If neither player is busted, the one with the highest-scoring hand wins
  - If both have the same score, it is a tie and no money changes hands

Q3b

Program specifications

- The blackjack program will allow a single player to play one hand of blackjack against the computer, starting with a fresh deck of cards
- It will have a simple text interface
  - But expandable into a GUI interface
- It will repeatedly display the state of the game and ask the Player whether (s)he wants a hit
- Once the Player says NO, the Dealer will play
- The results will be displayed

Initial design

- Similar to the top-level design of the Racquetball simulator from the textbook, we want to break up the blackjack algorithm into a few high-level tasks
- With one or two other people, quickly brainstorm what those tasks might be

Q4

Top-level algorithm for **Blackjack**

- Create initial card deck
- Deal initial cards
- Display game state
- Player plays until busted or chooses to stop
- Dealer plays until required to stop
- Report who wins

Top-level design, turned into code

Top-level design:

- Create initial card deck
- Deal initial cards
- Display game state
- Player plays until busted or chooses to stop
- Dealer plays until required to stop
- Report who wins

Write *main* based on the above. Write on your Quiz.

```
def main():
    deck = newDeck()
    player, dealer = initialDeal(deck)
    displayGame(player, dealer, False)
    playerPlays(player, dealer, deck)

    dealerPlays(player, dealer, deck)
    reportWinner(player, dealer)
    displayGame(player, dealer, True)
```

Top-level functions called by **main( )**

- newDeck()
  - Creates and returns a complete deck of cards
- initialDeal(deck)
  - deals cards from the deck to each player, returns the hands
- displayGameState(playerHand, dealerHand, showAll)
  - shows visible cards and player's scores. showAll is boolean
- playerPlays(playerHand, dealerHand, deck)
  - Allows player to choose hit or stay
- dealerPlays(playerHand, dealerHand, deck)
  - Dealer does hit or stay, based on the rules
- reportWinner(playerHand, dealerHand)
  - Determines and displays who wins.

Q5

Complete code for main()

```
def main():
    deck = newDeck()
    player, dealer = initialDeal(deck)
    displayGameState(player, dealer, False)
    playerPlays(player, dealer, deck)
    if handScore(player) > winningScore:
        print("BUSTED!  You lose.“)
    else:
        print("Now Dealer will play ...")
        dealerPlays(player, dealer, deck)
        reportWinner(player, dealer)
    displayGameState(player, dealer, True)
```

Summary of Loop Patterns

- The compute-in-a-loop pattern
- Six basic compute-in-a-loop patterns:
  - For loop
  - While loop
    - Interactive loop
    - Sentinel loop using a special value as the sentinel
    - Sentinel loop using no-input as the sentinel
  - Loop-and-a-half
    - Combined with use of no-input as the sentinel
  - File loop
  - Nested loops (this session)
  - Wait-for-event loop (this session)

Wait-for-event Loop Pattern

```
pre-loop computation
while [there is more data]:
        get data
        compute using the data
post-loop computation
```

```
pre-loop computation
while [the event has NOT occurred]:
        sleep for a bit
post-loop computation
```

Examine and run the
module3_waitForEventLoopPattern.py
module in the project you checked out today.

Nested Loops

- A *nested if* is an **if** inside an **if.**
- A *nested loop* is a loop inside a loop.
- Example:

```
for i in range(4):
    for j in range(3):
        print(i, j, i * j)
```

- What does it print?
  - Let's trace the **module1_multiplicationTables.py** module in the debugger
- What if we change the second range expression to

```
range(i + 1)
```

Q6-8, *turn in quiz*

Nested Loop Practice

- You will do several exercises that involve writing functions to generate patterned output.
  - In each, you will accumulate each line's output in a string, then print it.
  - Place this code inside **module2_nestedLoopPatterns.py** in today's project

Nested Loops – Class Exercise

- First, we will write a function to generate a pattern of asterisks like

```
**********
**********
**********
```

- We will write a function called `rectangleOfStars(rows, columns)`
- To produce the above pattern, we would call it with parameters **3** and **11**.

Nested Loop Practice – Your Turn

- Complete these definitions and test your functions
  - **triangleOfStars(n)** produces a triangular pattern of asterisks. For example, **triangleOfStars(6)** produces

```
*
**
***
****
*****
******
```

Hint: Use the same idea as the previous example. Start each line with an empty string. As you go through your inner loop, accumulate the line's characters. Print the line, then go on to the next iteration of the outer loop.

  - **triangleOfSameNum(n)** produces a triangular pattern of numbers. For example, **triangleOfSameNum(5)** produces

```
1
22
333
4444
55555
```

If you finish these exercises in class, continue with the remaining homework problems.