Lecture 03

Ready or Not

Control Flow Fundamentals: Boolean Expressions, Conditional Statements, and Loops

Go to poll.unc.edu

Sign-in via this website then go to **pollev.com/compunc**

VSCode: Open Project -> View Terminal -> npm run pull -> npm start

Announcements

- Videos 9-11 watch and take notes before class on Tuesday 1/21

- PS1 - Dwight's Beet Farm – Released tonight and Due Thursday 1/23
  - Your text needs to match the demo text _exactly_ to pass the grader. Every dot, capital letter, space, and so on.
  - After today's lecture you'll have the concepts needed for PS1.

Videos 6 through 8 Graded Warm-up Questions

- You may have out a single sheet, single page of handwritten notes
  - Why limit amount of notes? Thinking about what is most important aids learning.

- Complete the 9 questions in order
  - Caution: PollEv does not allow going back to change responses

- We will spend 5 minutes on this. When complete, please do not do anything to distract your neighbors.

- Open **pollev.com/compunc**

Challenge Question #0 - pollev.com/compunc

- Solve for yourself with paper/pencil then talk with your neighbors to see if you came to the same answer.
  - Don't use an interactive programming REPL!

- What values of a, b, and c would cause the following expression to evaluate to `true`?

```
((a && b) || c) && ((a || b) && !c)
```

Challenge Question #1 - pollev.com/compunc

- Solve for yourself with paper/pencil then talk with your neighbors to see if you came to the same answer.
  - Don't use an interactive programming REPL!

- What values of a, b, and c would cause the following expression to evaluate to `true`?

```
(a > 9) && (a < c) && (c < 12 && !b)
```

Challenge Question #2 - pollev.com/compunc
What is the *output* of these programs?

```
let x = 17;
if (x < 18) {
   print("A");
}

if (x > 13) {
   print("B");
} else {
   print("C");
}
```

…

```
let x = 17;
if (x < 18) {
      print("A");
} else {
   if (x > 13) {
      print("B");
   } else {
      print("C");
   }
}
```

Lecture Readiness - "Pulling" Class Materials

- When you come into lecture each day, the routine we'll get into is:

1. Open PollEv.com/compunc

2. Open VSCode -> View -> Terminal

3. In the Terminal, first run: **npm run pull**
   - This downloads the latest lecture materials.

4. Then run: **npm start**
   - This starts the development compiler and server allowing us to see the output of our code.

Magic 8 Ball

Generating Random Numbers

- The **`introcs`** Library has a special function for generating random numbers called... ***`random`***

- Before using `random`, we must import it into our program like **`print`**:

```
import { print, random } from "introcs";
```

- The random function generates a random number, so we can use it anywhere we can use a number:

```
let response: number = random(0, 2);
```

"Let response be a number variable that is assigned the result of calling the random function with the arguments 0 and 2."

- The two numbers we "give" to the random function specify the bounds of the random number generated (a number between 0 and 2, inclusive).

Hands-on: Magic 8-Ball

- Open: **8-ball-app.ts**

- Write a nested if-then-else statement (syntax below) at TODO #1 that will:

  if the **`response`** variable is equal to zero, **then** print "Very doubtful"
  otherwise,
  if `response` is equal to one, then print "Ask again later",
  otherwise, print "It is certain"

- **`if-then-else`** statement syntax:

```
if (<test>) {
     // then block
} else {
     if (<test>) {
         // then block
     } else {
         // else block
     }
}
```

- Check-in on pollev.com/compunc when your program prints one of these 3 messages

Repeating a Game

```
export let main = async () => {
   let isPlaying = true;
   while (isPlaying) {
      let question = await promptString("Ask a Yes/No Question");
      // ** response logic here **
      let shouldContinue = await promptString("Ask another? yes/no");
      // ** TODO **
   }
   print("Have a great day.");
};
```

Let's add a loop and a bit of extra logic.

Hands-on: Stopping the Loop

1. Notice the `while` loop's condition is the current value of **`isPlaying`**

2. Underneath the TODO, implement the following logic:

3. When `shouldContinue` is equal to "no", `isPlaying` should be assigned `false.` Otherwise, **`isPlaying`** should be remain **`true`**.

4. Save and test. You should be able to respond "no" and the game stops.

5. Check-in on PollEv.com/compunc and try to talk through *why* the loop stops with a neighbor.

Repeating a Game

```
export let main = async () => {
    let isPlaying = true;
    while (isPlaying) {
        let question = await promptString("Ask a yes / no question...");
        print(randomResponse());

        let shouldContinue = await promptString("Ask another? yes / no");
        if (shouldContinue === "no") {
            isPlaying = false;
        }
    }

    print("Have a great day.");
};
```

Pattern: Nesting **if-then** in an **else** Pattern

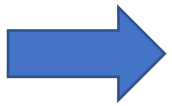
- It is commonly useful to nest additional if-then-else statements inside of subsequent else-blocks

- Why? It allows us to choose one next step from many possible options.
  - "If *this* then do X, otherwise if *that* do Y, *otherwise* do Z."

```
if (response === 0) {
    print("Very doubtful");
} else {
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
}
```

This is so common and useful, we tend to use simpler syntax for it…

```
if (response === 0) {
    print("Very doubtful");
} else {
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
}
```

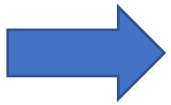


```
if (response === 0) {
    print("Very doubtful");
} else
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
```

1. First we remove the curly braces surrounding the if-then that is nested inside of the else-block.

This is so common and useful, we tend to use simpler syntax for it…

```
if (response === 0) {
    print("Very doubtful");
} else
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
```

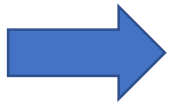


```
if (response === 0) {
    print("Very doubtful");
} else if (response === 1) {
    print("Ask again later");
} else {
    print("It is certain");
}
```

2. Then we clean up the spacing.

Using the **else-if** pattern is a change of *style* only. These two listings of code have the ***exact same logic***.

```
if (response === 0) {
    print("Very doubtful");
} else {
    if (response === 1) {
        print("Ask again later");
    } else {
        print("It is certain");
    }
}
```



```
if (response === 0) {
    print("Very doubtful");
} else if (response === 1) {
    print("Ask again later");
} else {
    print("It is certain");
}
```

Notice the code is visually simpler and cleaner by using else-if.

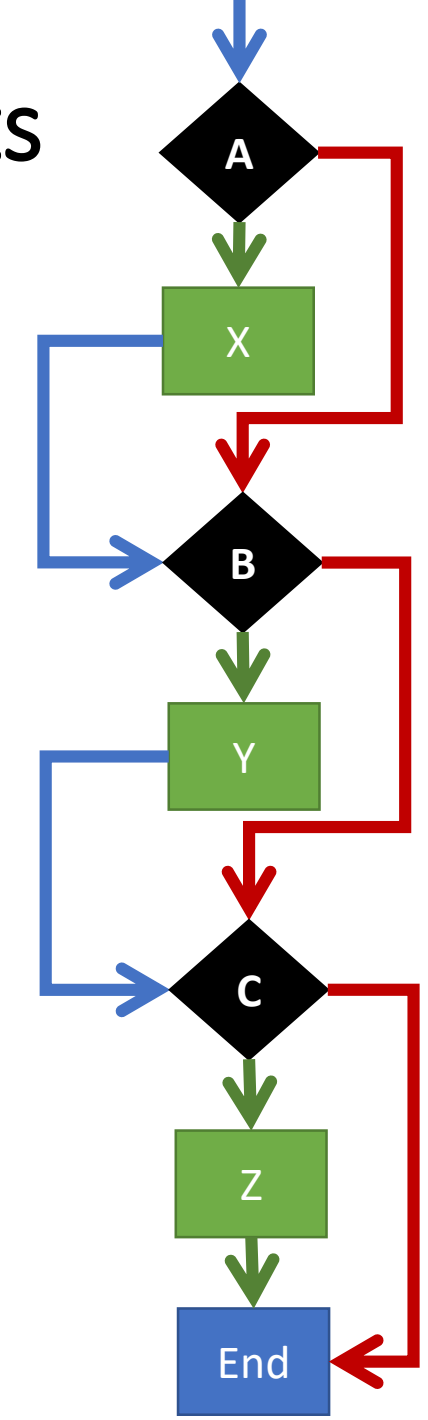
Follow-Along) Using the else-if Syntax Pattern

- Still in 8-ball-app.ts
- Reformat the conditional logic to use the else-if syntax pattern.
- Step 1) Remove the curly brace directly following the *first* else and its matching closing curly brace.
- Step 2) Clean up the spacing by bringing the nested if to directly follow else and unindenting.
- Check-in when complete! pollev.com/compunc

Many, independent `if-then-else` statements

- When two or more if-then-else statements are *not* nested, they are independent statements of one another.
- Each `boolean` test expression will be evaluated.
- Notice in the diagram that there is a path through *every* block X, Y, Z.

```
if (A) {
  print("X");
}

if (B) {
  print("Y");
}

if (C) {
  print("Z");
}

print("End");
```



Tracing through `else-if` statements

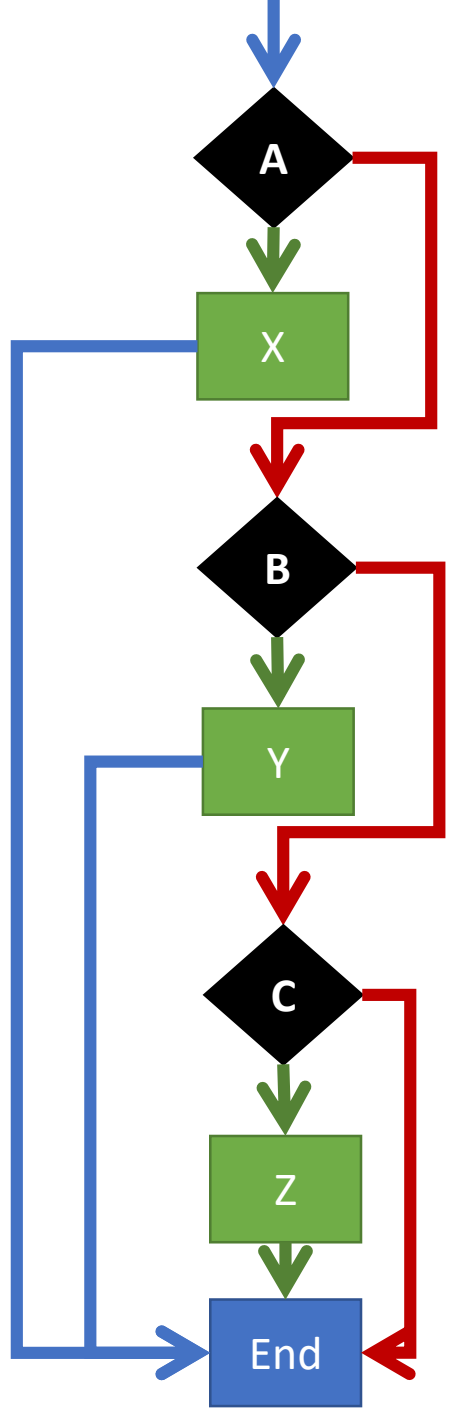
- The previous slide does not apply to else-if statements *because…*
  - An else-if is a nested if-then
  - It is nested in the else-block

- Each `boolean` test expression will be evaluated **until one evaluates to true**. The rest are then skipped.

- Notice in the diagram that there is a path through *only one* outcome X, Y, Z.

- Useful when there are many possible next steps but you only want to choose one.

```
if (A) {

    print("X");

} else if(B) {

    print("Y");

} else if(C) {

    print("Z");

}

print("End");
```



// Code Comments (1/2)

- We can add notes for ourselves in code using **comments**
- The computer ignores comments when running your program
- Single line comments begin with a //

```
// This is a single line comment
```

- Multi-line comments are surrounded by an opening **/*** and closing ***/**

```
/* This
   is
   a multi-line comment
*/
```

- Comments are also useful for ignoring code you've written without deleting it.

// Code Comments (2/2)

- In the early days of programming, we recommend writing comments liberally in your code to explaining what your program is doing in English.

- Comments are *free* so use them liberally.

- Comments are the easiest way to "take notes" when we are working on examples in lecture.

Statements (1/2)

- **Statements** are equivalent to an English sentence

- Statements *usually* end with a semi-colon ;
  - Like sentences end with a period!

- Each statement is an instruction you are giving to the computer.

```
print("hello, world");
```


That's a **statement**!

"Print 'hello, world' to the screen."

Statements (2/2)

- In a stored program, the computer will not carry out our instructions until we **run** the code
  - When you *write* programs, it is like you are writing a recipe down
  - When you *run* a programs, the computer is like the chef *following* your recipe
- Before the first statement runs, your program is a barren, empty world
  - *Your* code builds up its own little world piece-by-piece

```
print("hello, world");
```


That's a **statement**!

"Print 'hello, world' to the screen."

Move #4 – Function Calls

- The **function call** move is beautiful and magical. It's the power move.
- The computer *drops a bookmark* where the function call occurs and *jumps* into the function... *magic happens...*
- ...the computer then ***returns*** right to *the bookmark it dropped*, often with some data and the program continues on, business as usual.
- You've *already* used some function calls!
  - `print("hello, world");`