Conditional Execution

CSCI 185: Adding to our **control flow** repertoire

Announcements

- Tutorial 8 this Friday (3/31)
- Last day to submit HW5 (for a late penalty) is 4/3 (Sunday)
- Project Proposal due next Monday (4/3)
- Homework 6 posted (due in 2 weeks, 4/10)

Picking up where we left off on Wednesday

Functions Review: Challenge Problem

Remember our drawings from last Wednesday? Let's create a makeCreature function that allows us to position our creature anywhere on the screen

Open **01-make-creature**. Then, create a makeCreature() function that takes two arguments:

1. **x** – the x-coordinate where the creature should be centered
2. **y** – the y-coordinate where the creature should be centered

When I invoke the makeCreature function with different x and y arguments, the creature should be drawn in a different place on the screen (see 01-make-creature)

Challenge

Can you make your makeCreature function more flexible?

- What if you want the face to be different colors?
- What if you want it to be different sizes?

Outline

1. Control flow
2. Intro to conditional statements
    a. Why are they useful?
    b. How do they work?
3. Syntax
    a. if, else, and elif
4. Determining Truth
    a. Comparison operators
    b. Logical operators

Outline

1. **Control flow**
2. Intro to conditional statements
   a. Why are they useful?
   b. How do they work?
3. Syntax
   a. if, else, and elif
4. Determining Truth
   a. Comparison operators
   b. Logical operators

Control Flow: An Overview

- “Control Flow” refers to the order in which JavaScript statements and expressions are interpreted and executed by the browser.
- By default, JavaScript scripts execute from top to bottom. However, it is possible to define special blocks of code that can be repeated, skipped, or invoked on demand
- **Functions** are one example of this. Using functions, the same snippet of code can be invoked (upon request) with different data (arguments).

Control Flow: An Overview

This week and next week, we will be covering two new control mechanisms:

1. **Conditional Statements**
   a. Do something if a condition is met (and skip it otherwise)
2. **Loops (next week)**
   a. When a condition is True, keep doing something over and over again
   b. When the condition turns to False stop doing the thing

Outline

1. Control flow
2. **Intro to conditional statements**
    a. **Why are they useful?**
    b. **How do they work?**
3. Syntax
    a. if, else, and else if
4. Determining Truth
    a. Comparison operators
    b. Logical operators

Conditional Statements

Conditional statements are like a choose your own adventure novel:

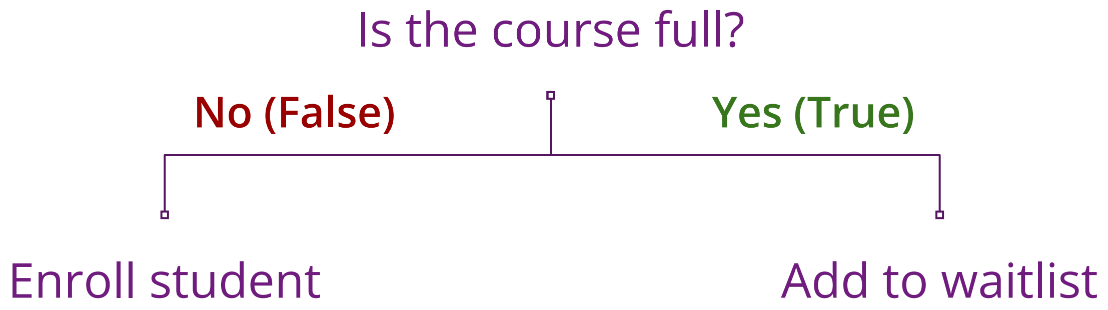
- If you choose to open Door #1, one thing happens, otherwise something else happens
- New choices can also be made as you step through new doors
- After a series of choices, many different outcomes become possible
- Each reader ends up in a different place: same book, different outcome

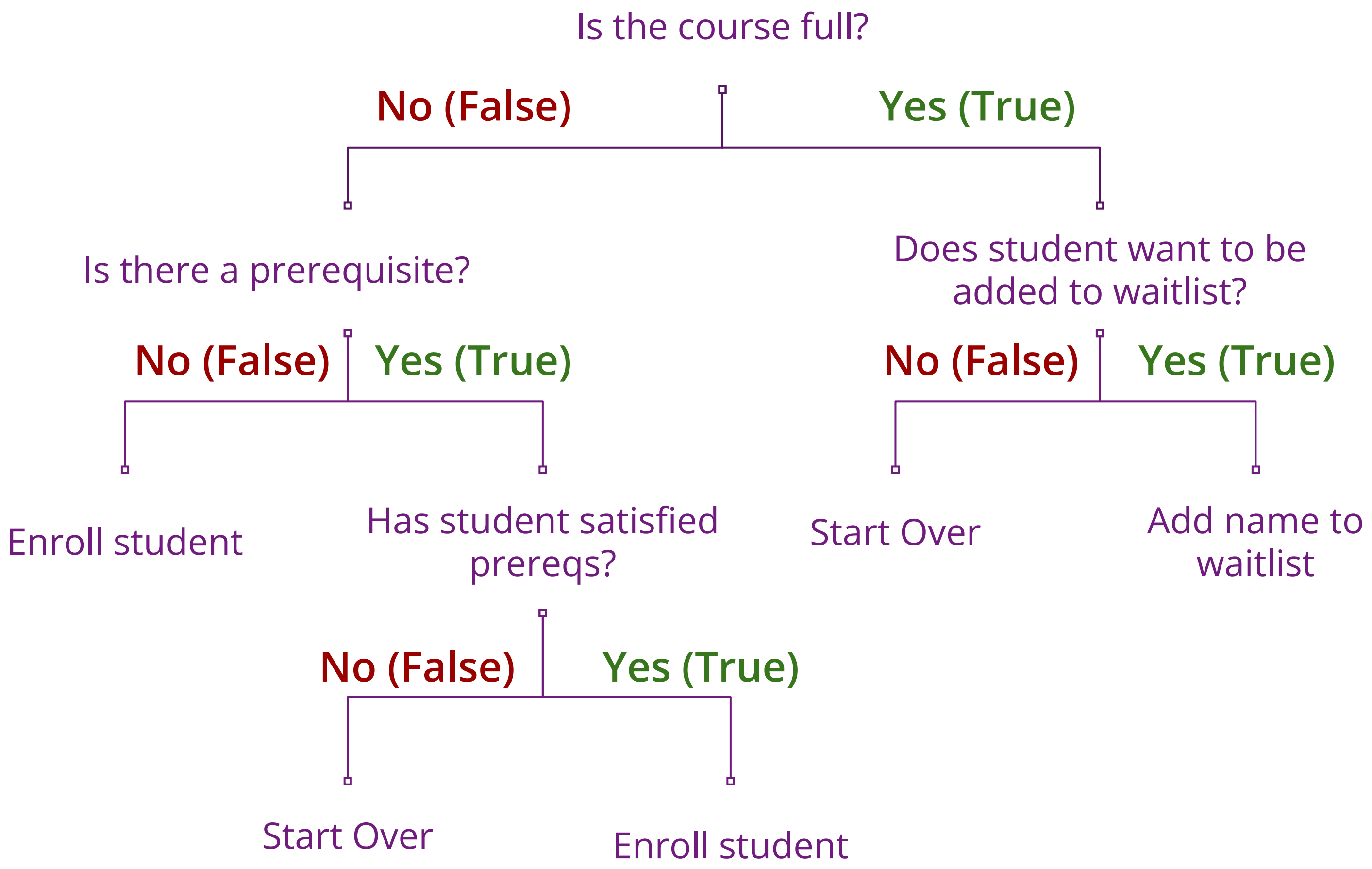
Computer programs work the same way. Depending on the data that's passed into the program, one part of your program will execute while another part gets skipped over

Conditional Statements

Conditional statements enable you to skip over some statements and execute others, depending on whether a condition is True or False.

EXAMPLE



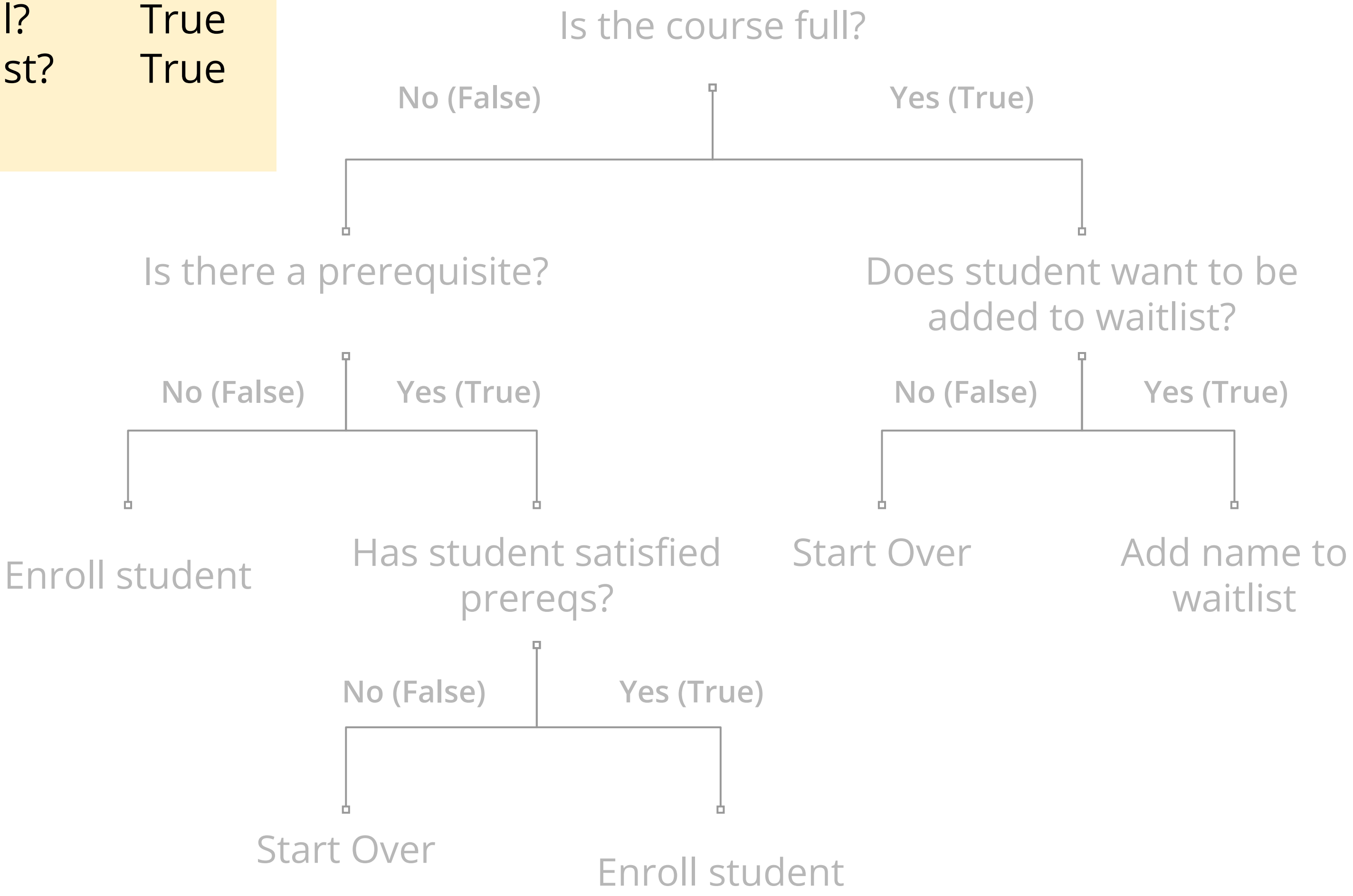

Is the course full?
No (False)
Yes (True)
Is there a prerequisite?
No (False)
Yes (True)
Does student want to be added to waitlist?
No (False)
Yes (True)
Enroll student
Has student satisfied prereqs?
Start Over
Add name to waitlist
No (False)
Yes (True)
Start Over
Enroll student

Everything needs to be put in terms of yes or no

```
Is the course full?
    |—— No (False)
    |    |—— Is there a prerequisite?
    |         |—— No (False)
    |         |    └—— Enroll student
    |         |—— Yes (True)
    |              |—— Has student satisfied prereqs?
    |                   |—— ...
    |—— Yes (True)
         |—— Does student want to be added to waitlist?
              |—— No (False)
              |    |—— Start Over
              |—— Yes (True)
                   |—— Add name to waitlist
```

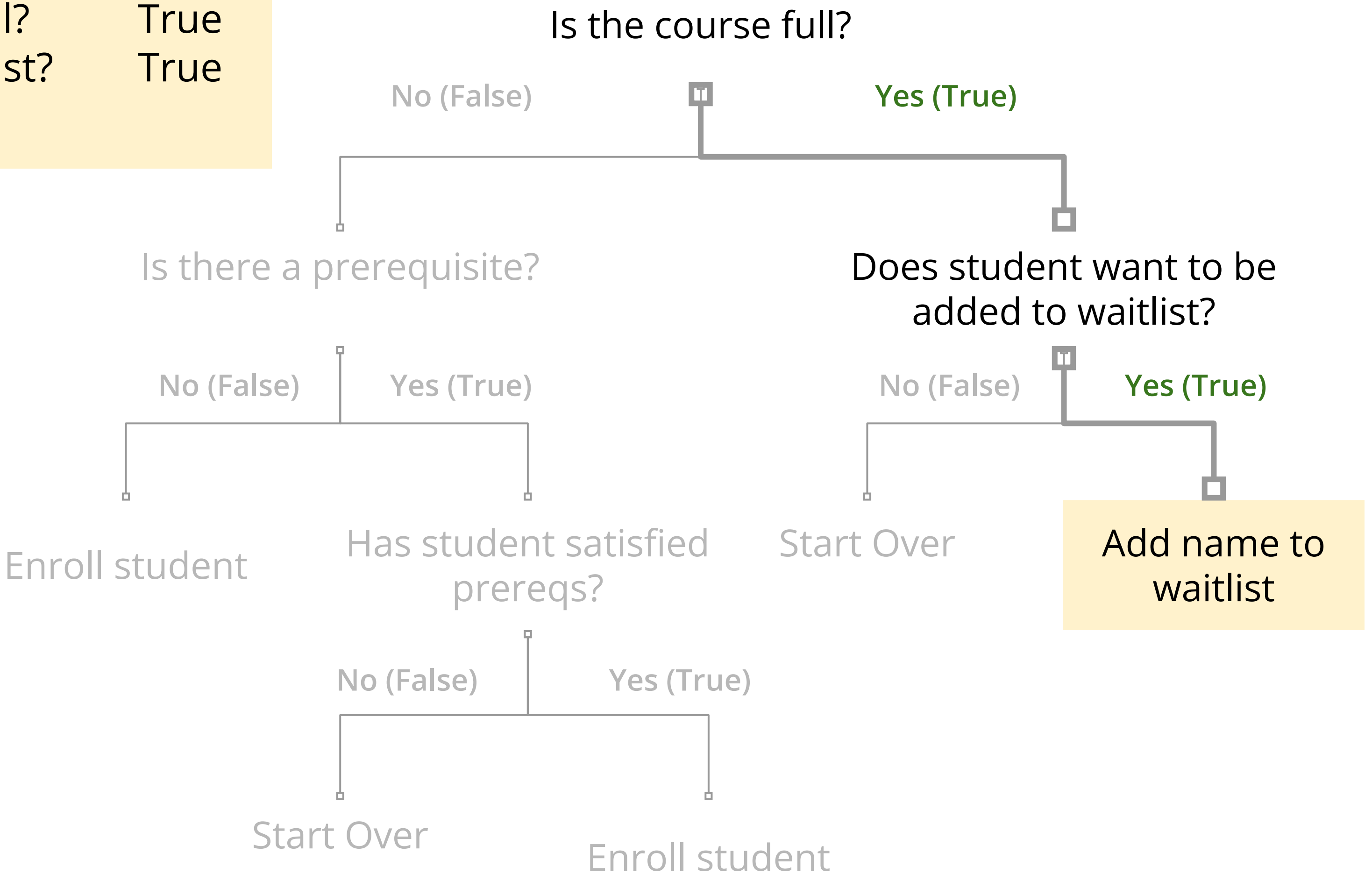
Data (use case):

1.	Is course full?	True
2.	Wants waitlist?	True



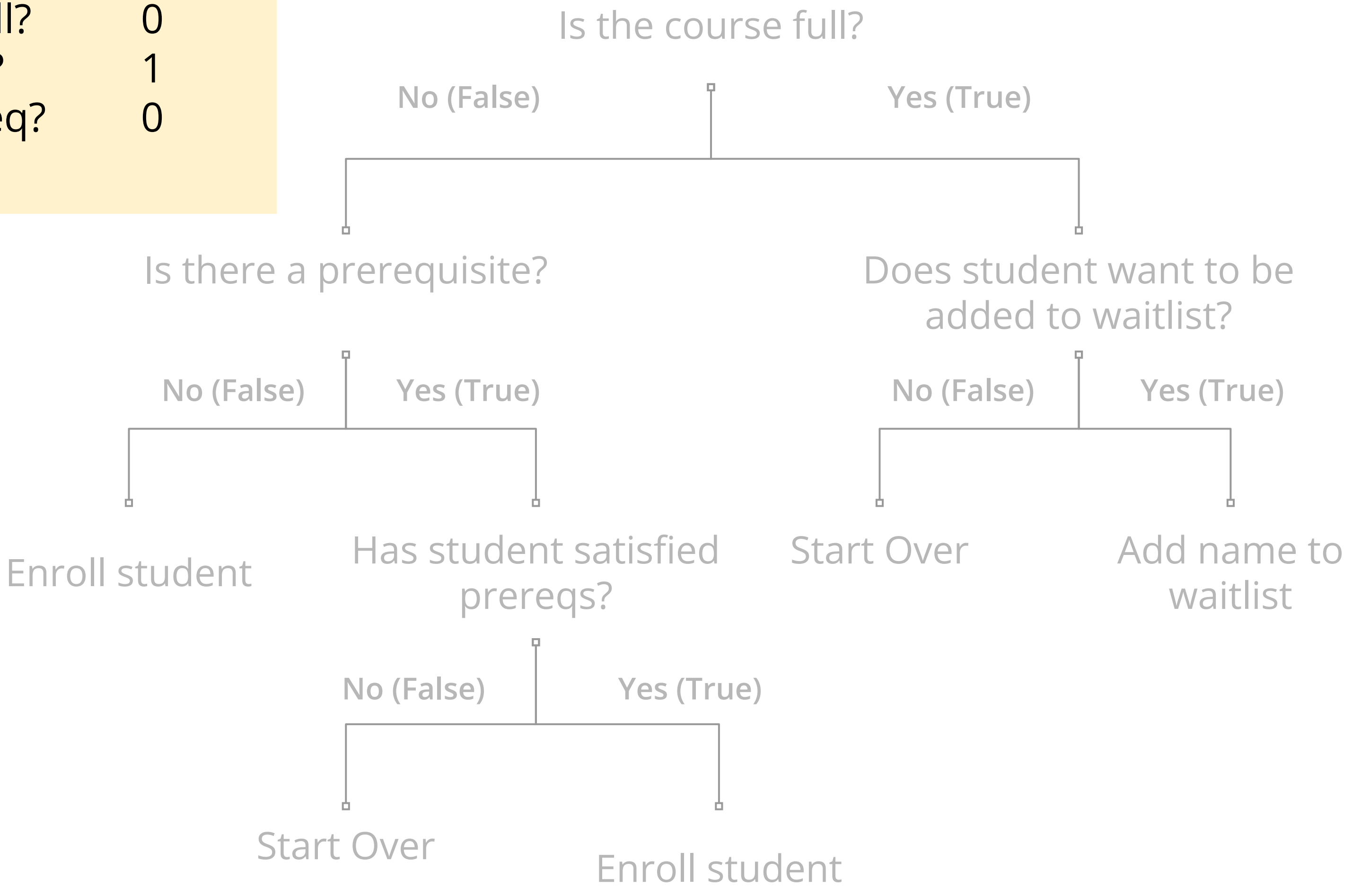
Data (use case):

1.	Is course full?	True
2.	Wants waitlist?	True



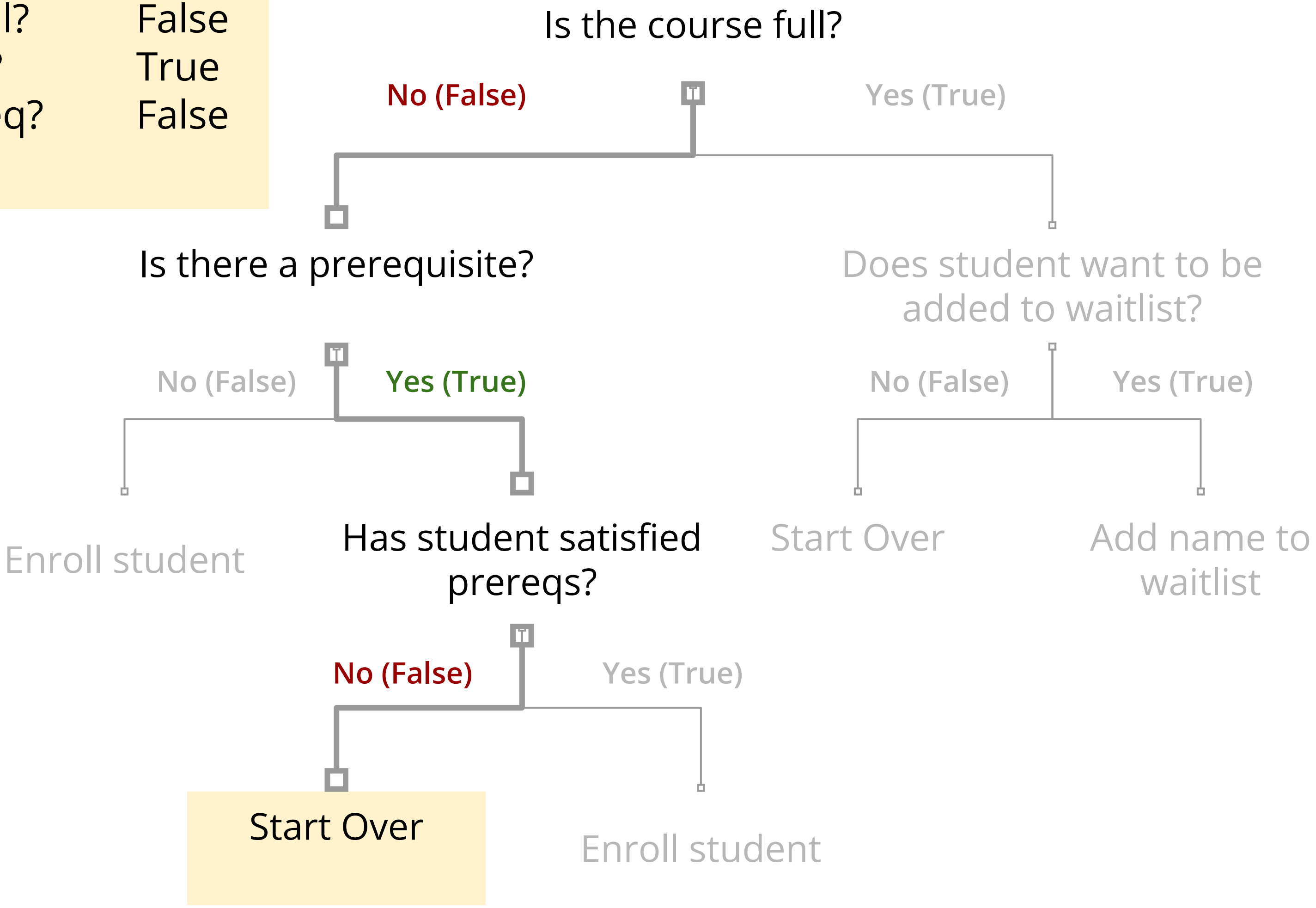
Data (use case):

1.	Is course full?	0
2.	Has prereq?	1
3.	Taken prereq?	0



Data (use case):

1.	Is course full?	False
2.	Has prereq?	True
3.	Taken prereq?	False



Yes / No questions can be nested

As seen in the previous example, conditional statements can be nested:

- If the course is already full, it may not make sense go through the process of checking whether the student has fulfilled the prerequisites to register (although it may – validating this assumption would be important)
- Ditto if the student doesn’t want to be added to the waitlist.

Outline

1. Control flow
2. Intro to conditional statements
   a. Why are they useful?
   b. How do they work?
3. Syntax
   a. if, else, and elif
4. Determining Truth
   a. Comparison operators
   b. Logical operators
   c. Truth tables

If Statement

```
if (condition) {
    statement 1
    statement 2
    ...
}
```

CONDITION
Boolean expression that evaluates to True or False.

BLOCK
If the condition evaluates to True, the block executes. Otherwise, the block is skipped.

If Statement: Example (Leap Year)

```
let year = 2023;
...
console.log('February 26');
console.log('February 27');
console.log('February 28');
if (year % 4 == 0) {
    console.log('February 29');
}
console.log('March 1');
```

Expression that evaluates to either True or False

2019: This condition does not execute

2020: This condition does execute because 2020 is a leap year (and 4 divides evenly into 2020)

Comparison Operators

Comparison operators compare two operands according to a comparison rule and evaluate to either True or False (boolean)

Operator	Description
===	Strict Equality. Both values and data types are equal.
==	Value Equality: If the values of two operands are equal, then the condition becomes true.
!=	If values of two operands are not equal, then condition becomes true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.

Comparison Operator

```
let year = 2023
...
if (year % 4 === 0) {
        console.log('February 29');
}
```

Comparison Operator

Evaluates to **False** (so the code block does not execute)

```
let year = 2023
...
if (3 === 0) {
        console.log('February 29');
}
```

If / Else Statement

```
if (condition) {
    statement 1
    statement 2
    ...
} else {
    statement 3
    statement 4
    ...
}
```

CONDITION
Boolean expression that evaluates to True or False.

BLOCK
Executes if the condition evaluates to True.

BLOCK
Executes if the condition evaluates to False

Only one of the blocks will execute. They never both execute. It's an either / or situation.

If / Else Example: Even or Odd?

If / Else statements are useful for scenarios that are binary (yes or no). Example:

- either the class is full or it isn’t
- either you’re a sophomore at UNCA or you’re not

Practice:

1. Write a function called **evenOrOdd** that takes 1 parameter — an integer called **num** — and returns a string that says either “even” or “odd”
2. Prove that your function works by printing the results of several function calls to the screen

If / Else Example: Even or Odd [ANSWER]?

```
function evenOrOdd(num) {
    if (num % 2 == 0) {
        // if you divide by 2 and the remained is 0, it's even
         return 'even';
    } else { // otherwise, it's odd
         return 'odd';
    }
}

console.log('5:', evenOrOdd(5));
console.log('18:', evenOrOdd(18));
console.log('125:', evenOrOdd(125));
```

02_even_or_odd.js

Conditionals Summary

1. **If** Statement

```
if (condition) {
    statement 1
    statement 2
    ...
}
```

CONDITION
Boolean expression that evaluates to True or False.

BLOCK
If the condition evaluates to True, the block executes. Otherwise, the block is skipped.

```
console.log('February 28');
if (year % 4 == 0) {
    console.log('February 29');
}
console.log('March 1');
```

"If I divide the year by 4 and the remainder is 0, it's a leap year, so print February 29."

If / Else Statement

```
if (condition) {
    statement 1
    statement 2
    ...
} else {
    statement 3
    statement 4
    ...
}
```

CONDITION
Boolean expression that evaluates to True or False.

BLOCK
Executes if the condition evaluates to True.

BLOCK
Executes if the condition evaluates to False

Only one of the blocks will execute. They never both execute. It's an either / or situation.

2. If / Else Statement

Useful for either / or conditions:

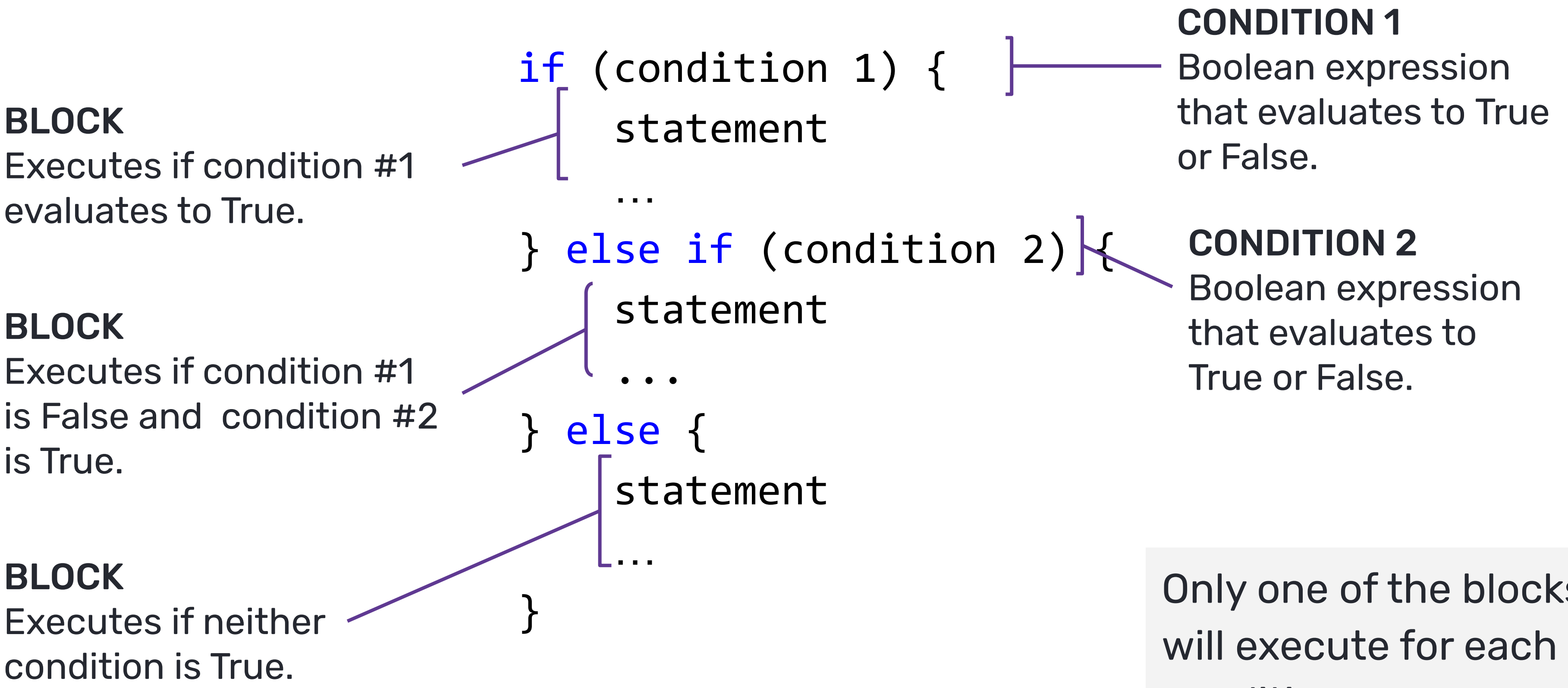
```
const evenOrOdd = (num) => {
    if (num % 2 == 0) {
        // if you divide by 2 and the [illegible] 0, it's even
        return 'even';
    } else { // otherwise, it's odd
        return 'odd';
    }
}
```

"If I divide num by 2 and the remainder is 0, it's an even number."

"If num is not divisible by 2 the if condition is False, so num must be odd."

Practice: Light Bulb Activity
03-light-bulb

3. **If / Else if / Else** Statement



```
if (condition 1) {
    statement
    ...
} else if (condition 2) {
    statement
    ...
} else {
    statement
    ...
}
```

BLOCK
Executes if condition #1 evaluates to True.

BLOCK
Executes if condition #1 is False and condition #2 is True.

BLOCK
Executes if neither condition is True.

CONDITION 1
Boolean expression that evaluates to True or False.

CONDITION 2
Boolean expression that evaluates to True or False.

Only one of the blocks will execute for each condition.

Practice: Game Controller

04-game-controller

Outline

1. Control flow
2. Intro to conditional statements
    a. Why are they useful?
    b. How do they work?
3. Syntax
    a. if, else, and elif
4. **Determining Truth**
    a. **Comparison operators**
    b. **Logical operators**

Logical Operators

and (&&), or (||), and not (!)

Sometimes you want to check two or more things...

- Sometimes you want to check if more than one thing is true or false
- And sometimes you want to check if something is NOT true or NOT false

Examples:

- Is the student a second year and a history major?
- Is the student not graduating this year?

For these kinds of questions, you need to understand how to use logical operators...

Logical Operators

Logical operators provide additional ways to determine whether something is true or false:

Operator	**Description**
&&	If both operands are true then the expression evaluates to true. Otherwise, the expression evaluates to false
||	If either or both operands are true then the expression evaluates to true. If both operands are false, the expression evaluates to false
!	If the operand is false than the expression evaluates to true (and vice versa)

AND, OR, & NOT

Fundamentally, a computer is checking whether two switches are turned on and off:

- true (on) → same as 1
- false (off) → same as 0

Depending on which ‘switches’ are turned on and off (in combination), different things happen

Example: AND

```
function inBetween(num, high, low) {
	return high > num && num > low;
}

console.log(inBetween(90, 70, 20))

console.log(inBetween(30, 70, 20))
```

OUTPUT:

```
false
true
```

Example: OR

```
function giveMovieDiscount(isStudent, isSenior, isChild) {
	return isStudent || isSenior || isChild;
}

console.log(giveMovieDiscount(false, false, false));
console.log(giveMovieDiscount(false, true, false));
console.log(giveMovieDiscount(true, false, false));
```

OUTPUT:

```
false
true
true
```

Practice: Color Mixer

05-color-mixer