9/26/22

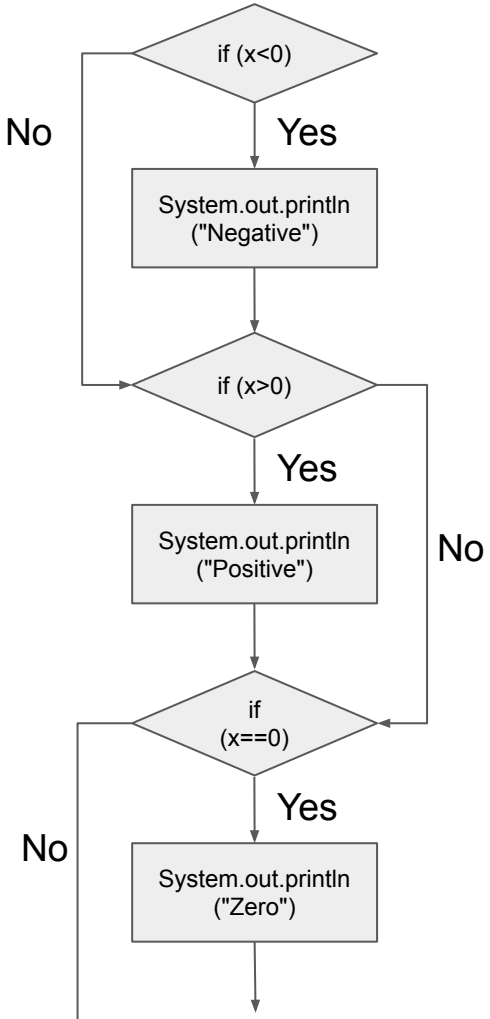
3.4
else-if review

Code without else-if

```java
import java.util.Scanner;

class Main {
  // Write a program that asks the user for an integer.
  // Print "The number is positive." or
  //       "The number is negative." or
  //       "The number is zero."

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter a number.");
    int x = scanner.nextInt();
    if (x < 0) {
      System.out.println("The number is negative.");
    }
    if (x > 0) {
      System.out.println("The number is positive.");
    }
    if (x == 0) {
      System.out.println("The number is zero.");
    }
  }
}
```

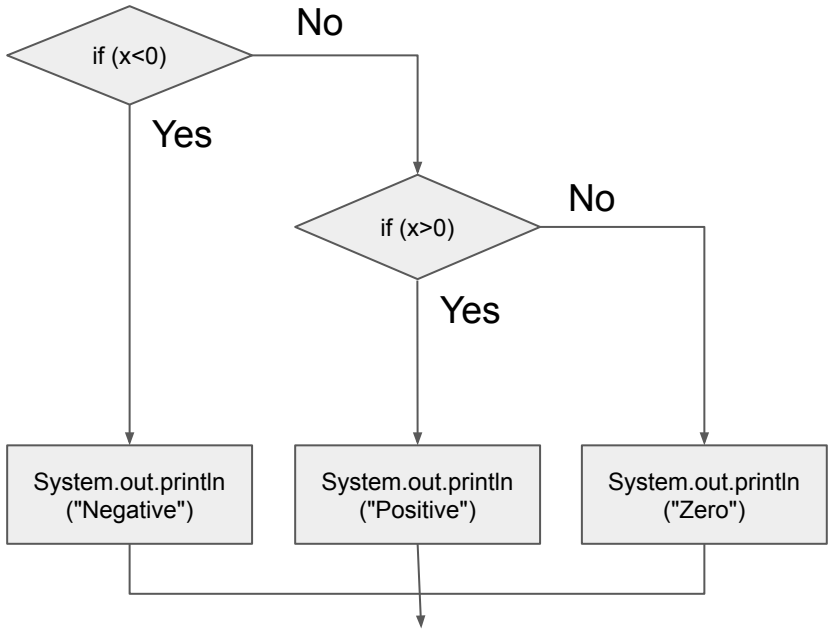


Code with else-if is faster

```java
import java.util.Scanner;

class Main {
  // Write a program that asks the user for an integer
  // Print "The number is positive." or
  //       "The number is negative." or
  //       "The number is zero."

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter a number.");
    int x = scanner.nextInt();
    if (x < 0) {
      System.out.println("The number is negative.");
    } else if (x > 0) {
      System.out.println("The number is positive.");
    } else {
      System.out.println("The number is zero.");
    }
  }
}
```



- Jumps around remaining conditions
- Can use information learned in previous cases – no longer need to check for zero explicitly, since number is known to not be positive or negative

else if

In each `else if`, we still know what we learned from previous conditions.

`if (score >= 80)` ← if this is true, we don't need to also check that score<90.

We know it must be, or the `if (score >= 90)` case would've already executed and skipped the rest.

In the final `else`, we know that score<60 and don't have to check.

```java
import java.util.Scanner;

class Main {
  // Write a program that asks the user for a test score from 0-100.
  // Print the letter grade for the given score.
  // (A-, B+, etc. modifiers not required, but you can if you want.)
  //
  // A    90-100
  // B    80-89
  // C    70-79
  // D    60-69
  // F    59 and under

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter your score.");
    int score = scanner.nextInt();
    if (score >= 90) {
      System.out.println("You got an A!");
    } else if (score >= 80) {
      System.out.println("You got an B!");
    } else if (score >= 70) {
      System.out.println("You got a C!");
    } else if (score >= 60) {
      System.out.println("You got a D!");
    } else {
      System.out.println("You got a F!");
    }
  }
}
```

3.5

Compound Boolean Expressions

Logical Operators

Logical And

p `&&` ***q***

Evaluates boolean expressions ***p*** and ***q***.

Evaluates to `true` if ***p*** and ***q*** are both `true`, `false` otherwise.

```
if (sunny && warm) {
    …
}
```

Logical Or

p `||` ***q***

Evaluates boolean expressions ***x*** and ***y***.

Evaluates to `true` if ***p*** or ***q*** are `true`, `false` otherwise.

```
if (christmas || halloween)
{
    …
}
```

Logical Not

`!`***p***

Evaluates boolean expression ***p***.

Evaluates to `true` if ***p*** is `false`.

Evaluates to `false` if ***p*** is `true`.

```
if (!day.equals("Sunday"))
{
    …
}
```

Why p and q? In logic textbooks, the "default" names for logical propositions are p and q.

Logical Or is inclusive or

Logical Or
p `||` ***q***
Evaluates boolean expressions ***p*** and ***q***.
Evaluates to `true` if ***p*** or ***q*** are `true`, `false` otherwise.

`if (christmas || halloween)`
`{`
…
`}`

Exclusive Or
In English, "or" is often **exclusive or**, choosing between two possibilities.
"Do you want to be Player 1 or Player 2 in this game?"

Inclusive Or
"He ate cake, or ice cream, or both."
Either proposition can be true for the whole statement to be true.

The `||` operator performs an **inclusive or**.
Either side can be true for it to be true.

```
if (ateCake || ateIceCream) {
      … executes if either is true …
}
```

Why is it double &&, double ||?

This dates back to the C programming language (1970) that inspired Java and many other languages.

`&` and `|` are "bitwise" operators that "twiddle" individual bits in integers, like we saw with the binary addition repl.it exercise.

`&` and `|` were taken, so C used `&&` and `||` for logical operators.

That made its way into many other languages, including Java, JavaScript, Ruby, Scala, Go, …

But not Python! Python spells out `and` and `or` !

C/C++/Java use symbols for most operators instead of words.

For this course, you don't need to know `&` and `|` . Just remember to use `&&` and `||` instead!

Why is it ! for logical negation?

- This dates back even before C, to the B programming language! (1969)
- Python's equivalent operator is the word `not`
- In logic, the symbol "¬" is often used to represent negation… but it is hard to type, and was even harder to type back in 1969! Keyboards usually have an exclamation point, though!
- As Wikipedia says, people pronounce `!` as "bang" or "not"

Various notations to represent logical negation (Wikipedia)

Notation	Plain Text	Vocalization
$\neg p$	¬p	Not *p*
$\sim p$	~p	Not *p*
$-p$	-p	Not *p*
N*p*		En *p*
p'	p'	*p* prime, *p* complement
$\bar{p}$	¯p	*p* bar, Bar *p*
$!p$	!p	Bang *p* Not *p*

Truth Table - &&

p	q	p && q
true	true	true
true	false	false
false	true	false
false	false	false

Truth tables are a useful tool for mapping out the possible inputs and outputs of boolean expressions. This one is pretty simple, but truth tables can be very useful for understanding how complex boolean expressions behave.

Truth Table - ||

You Decide!

p	q	p || q
true	true	
true	false	
false	true	
false	false	

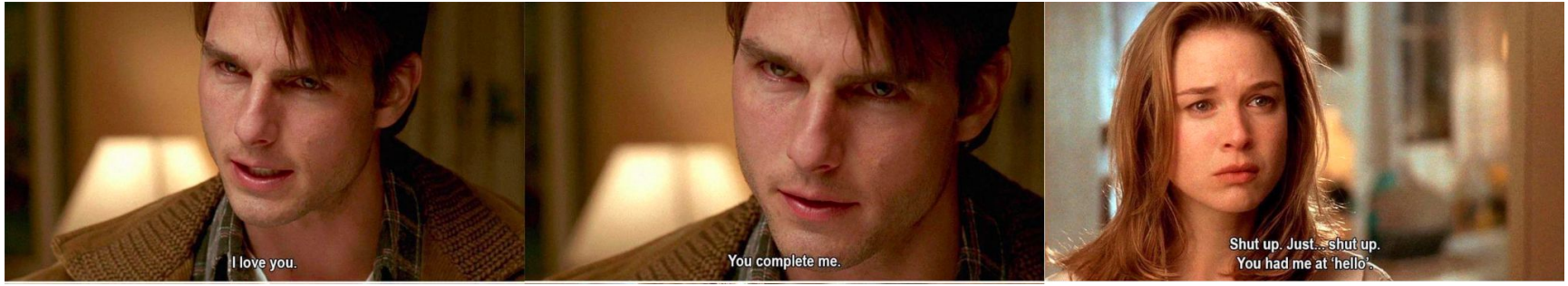
Truth Table - !

You Decide!

p	**!p**
`true`	
`false`	

Short-Circuit Evaluation

When evaluating `&&` and `||` expressions, Java will make its decision as early as possible.



Short-Circuit Evaluation with &&

Main.java

```java
import java.util.Scanner;

class Main {
  private static boolean readBoolean(String name) {
    System.out.println("Enter a value for " + name + ", either true or false.");
    Scanner scanner = new Scanner(System.in);
    return scanner.nextBoolean();
  }

  public static void main(String[] args) {
    boolean result = readBoolean("p") && readBoolean("q");
    System.out.println("The result of p && q == " + result);
  }
}
```

Console × Shell ×

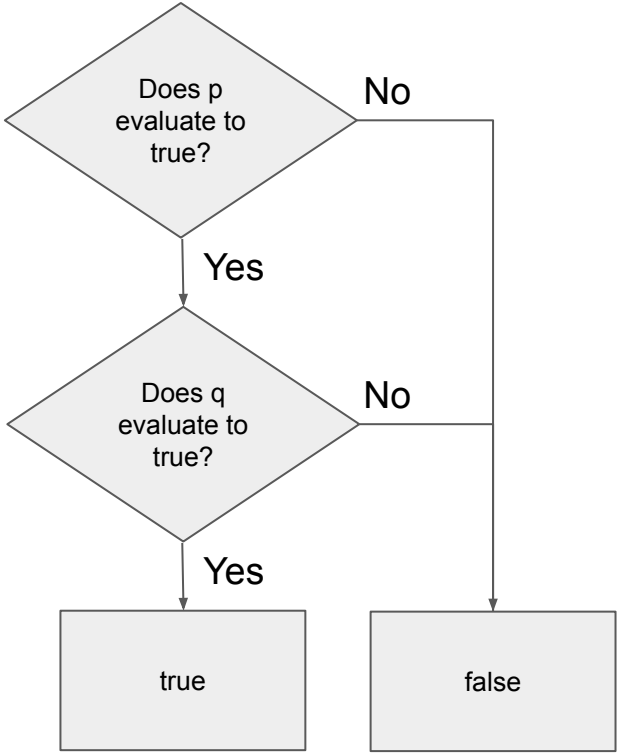
```
> sh -c javac -classpath .:target/dependency/* -d . $(find . -type f -name '*.java')
> java -classpath .:target/dependency/* Main
Enter a value for p, either true or false.
true
Enter a value for q, either true or false.
true
The result of p && q == true
> java -classpath .:target/dependency/* Main
Enter a value for p, either true or false.
false
The result of p && q == false
>
```

How Java evaluates `p && q`:

1. Evaluate `p`. If `p` is `false`, stop and return `false`.
2. Evaluate `q`. If `q` is `true`, return `true`, otherwise, return `false`.

Short-Circuit Evaluation with &&

```
p && q
```



Short-Circuit Evaluation with ||

Main.java

```java
import java.util.Scanner;

class Main {
  private static boolean readBoolean(String name) {
    System.out.println("Enter a value for " + name + ", either true or false.");
    Scanner scanner = new Scanner(System.in);
    return scanner.nextBoolean();
  }

  public static void main(String[] args) {
    boolean result = readBoolean("p") || readBoolean("q");
    System.out.println("The result of p || q == " + result);
  }
}
```

Console × Shell ×

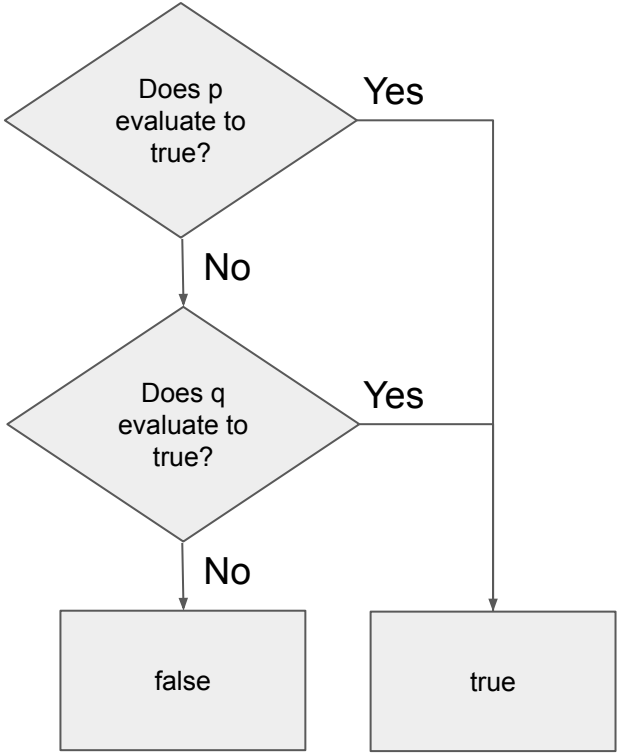
```
> sh -c javac -classpath .:target/dependency/* -d .
nd . -type f -name '*.java')
> java -classpath .:target/dependency/* Main
Enter a value for p, either true or false.
true
The result of p || q == true
> java -classpath .:target/dependency/* Main
Enter a value for p, either true or false.
false
Enter a value for q, either true or false.
true
The result of p || q == true
>
```

How Java evaluates `p || q`:

1. Evaluate `p`. If `p` is `true`, stop and return `true`.
2. Evaluate `q`. If `q` is `true`, return `true`, otherwise, return `false`.

Short-Circuit Evaluation with ||

```
p || q
```



Practice!

Repl.it for today:

Compound Boolean Expressions 1

Compound Boolean Expressions 2

cafeteria